パソコンプログラマのためのクリエイティブマガジン

平成3年4月2日第三種郵便物認可　1991年9月1日発行(毎月1回18日) 第1巻第9号通巻9号

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1991 9 SEPTEMBER

定価980円

特集

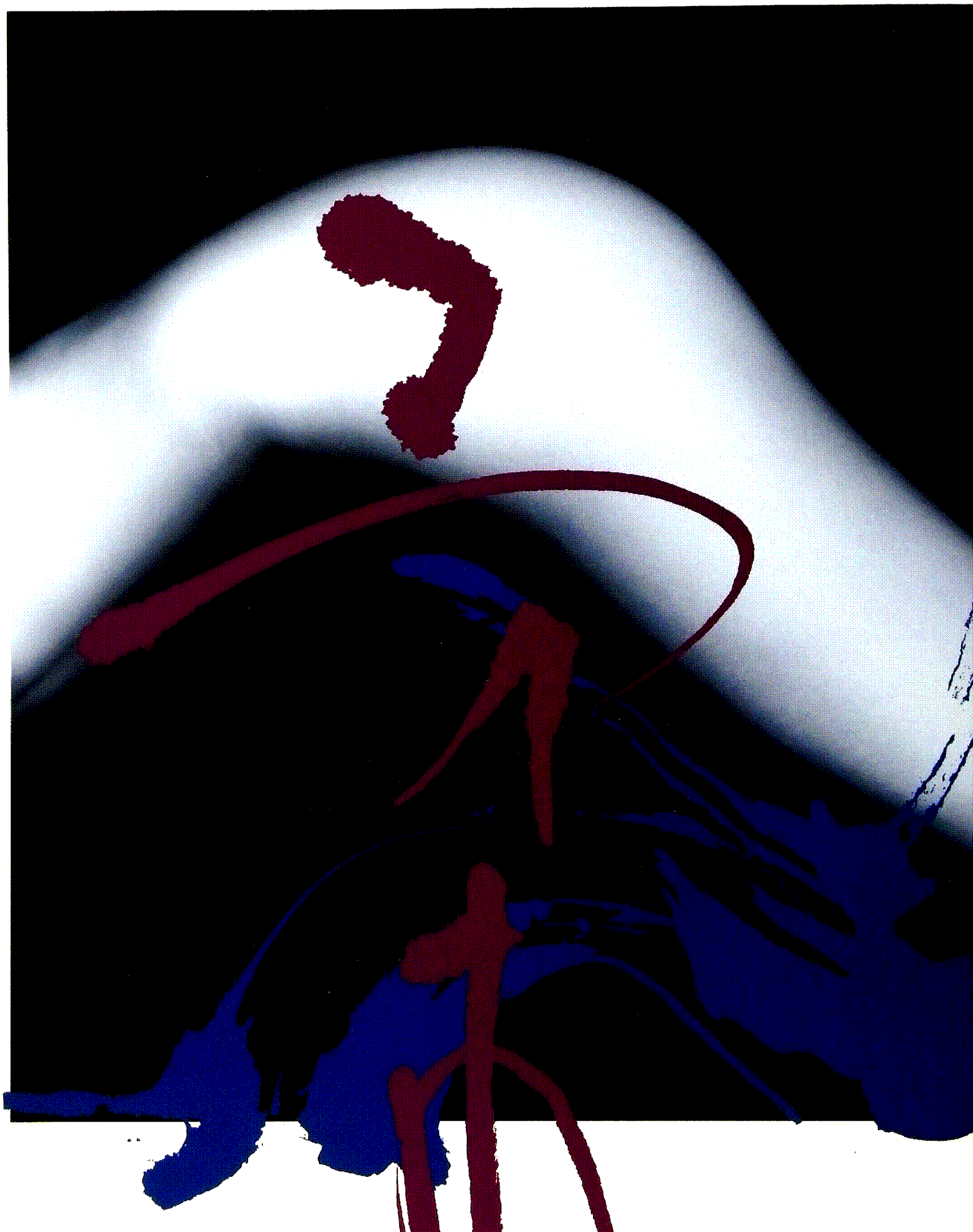
グラフィックス プログラミングの世界

基礎からレイトレーシングまで，プログラミング技法を徹底解説

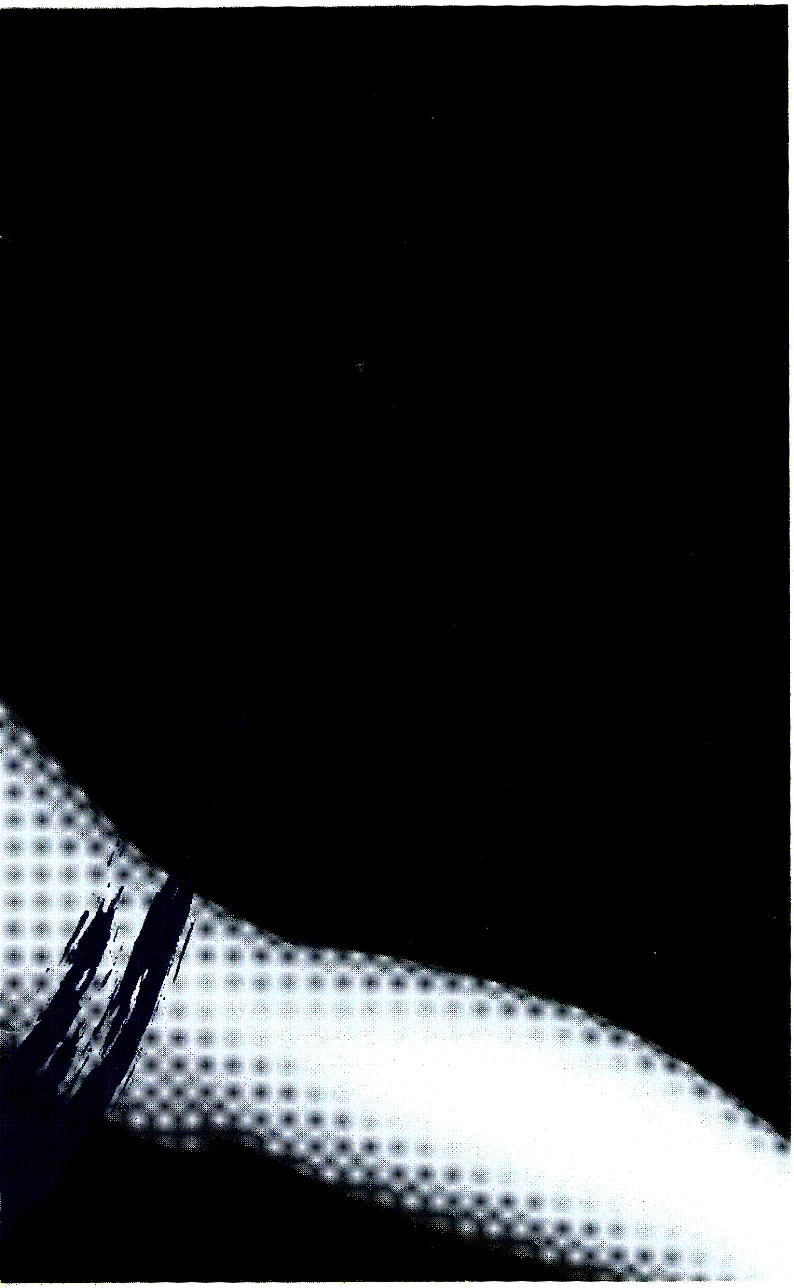
付録:5"2HDフロッピーディスク

本誌掲載プログラム／綴じ込み付録テキストファイル(PC-9801シリーズ・グラフィックBIOS/LIO機能一覧)／クイズ・ランダムスクリーン解読ツール／データファイル実行ツール「えぐせぐぜ」／PC-9801用グラフィックスライブラリ「GR.LIB」

SE
Shoeisha Co., Ltd.



柔らかさを追求する
パーソナルコンピュータはAXです。



世界の標準機を、日本標準機に。我々が、パソコンの新しい時代を拓きます。

Acer
Canon
CASIO
HITACHI
KYOCERA
MITSUBISHI
NCR
OKI
olivetti
SANYO
SHARP
SONY
UNISYS
YHP

アルファベット順

柔軟な拡張性を追求するなら、AXパソコンです。単にソフトウェアを搭載するだけでなく、拡張ボードを加えたり、コンピュータ自体をパワーアップする。AXパソコンを使うことにより、環境そのものを自由に構築することができます。さらに、DVI、CD-ROM、メモリ、スキャナ、FAX、ボイスメール等を導入することにより、次世代パソコンのあるべき姿を形成します。
AXパソコンは事実上の世界標準機となっているIBM PC/ATと、その互換機の世界をベースにしたパソコンの共通使用。世界で生まれた最新技術を素早く取り入れることもできるのです。メーカーが違えば互換性のなかった、これまでのパソコンの世界。そこに、「パソコンはもっと使いやすくなくては」というユーザー本位の発想を持ちこんだのが、AXです。メーカー間の、そして国際間の壁を取り払い、パソコンの新しい世界を築くために。AX協議会会員585社('91年4月現在)は、それぞれの技術を結集しているのです。

THE ARCHITECTURE EXTENDED—AX

お問い合わせ先：株式会社 翔泳社
TEL.03-3263-0447

〈資料請求番号P02〉

HIGH C386

NEW

数々の実績と定評を得たプロテクトモード対応C言語コンパイラが、ver1.7にバージョンアップされました。新たな機能と、処理速度の向上を持って、C言語のプロフェッショナル技術者にお届けします。

新たな機能

1. 80386/Weitek3167の他に、80486内蔵FPU、Weitek4167、Cyrix社EMC87コプロセッサをサポート。
2. マイクロソフト社のMASM及びOS/2 ver2.0のOMFと同じフォーマットのオブジェクトを生成可能。
3. 従来のライブラリに、割り込み関数、ファイル管理関数、シグナル機能、子プロセス関数を強化して追加。
4. 従来品よりも多種のコンパイラトグルスイッチと、コンパイラドライバのオプションスイッチを装備。
5. よく使われる重要な項目を本マニュアルより抜き出した、Quick Reference Manualを追加。

強化された関数群

割り込み関数
- _getpvect　現在のプロテクトモードにおける割り込みハンドラの取得
- _getrvect　現在のリアルモードにおける割り込みハンドラの取得
- _setpvect　プロテクトモードにおける割り込みハンドラのセット
- _getrvect　リアルモードにおける割り込みハンドラのセット
- _setrpvect　プロテクト／リアルモードにおける割り込みハンドラのセット

ファイル管理関数
- _fdopen　openされたファイルハンドルとストリームを結び付ける
- _faopen　ファイルをopenするとともに、分割モードをセットする

シグナル関数
- _Kill　シグナルを送る
- raise　シグナルを発する
- signal　シグナルハンドラの設定を行なう

子プロセス関数
- _exec＊　子プロセス利用のための関数群

動作環境

必要なハードウェア
対応機種
NEC 32bit機対応版
- NEC　PC-9800シリーズ(XL2, RA, RL, LS, ES, RA21/51, RS21/51, T, DA, DS, NS)
- NEC　PC-H98シリーズ(Model 60, 70, 100)
- EPSON　PC-386シリーズ(386, LS, V, M, G)

IBM PC/AT互換機対応版
- 東芝　J-3100(SGT, SGX, GXS), J-3300
- IBM　PC/AT互換機, 各種AXマシン(80386/SX搭載機)

富士通 32bit機対応版
- 富士通　FMR-50(HV, S2, TX), FMR-70(HXI, HX2, HX3)

ハードディスク
1MByte以上のプロテクトモードメモリ

必要なソフトウェア
MS-DOS

NEC, EPSON機	MS-DOS	ver3.3以上
IBM PC/AT機, AX機	PC-DOS	ver3.2以上
東芝 J-3100等	日本語DOS	ver3.1
	英語DOS	ver3.3
富士通 FMR	MS-DOS	ver3.1 Level28以上

Phar Lap Software Inc. 386|TOOLBOX(弊社取り扱い製品)

定価 ¥178,000

ベンチマークテスト

WHETSTONE(自社計測)
使用機器：PC-9801RL20MHz
＋数値演算コプロセッサ(80387 or Turbo-3167)

emulator ＋ noFPU	58	[K whetstone/sec]
emulator ＋ 80387	200	[K whetstone/sec]
80387 inline code	1540	[K whetstone/sec]
Turbo-3167 inline code	2380	[K whetstone/sec]

DHRYSTONE(ソフトバンク㈱刊 月刊Cマガジン'90.8月号より抜粋)
使用機器：PC-9801RA2 数値演算コプロセッサ(80387)

HIGH C386 6250 [dhrystone/sec] 8(SEC)
for 50000 PASSES

4GByteのメモリが使えるC言語
新バージョン！ HIGH C386 ver1.7

必須

386|TOOLBOX

MS-DOSの640KByteという狭いメモリの制限を越えて、4GByteという広大なアドレス空間を利用者に提供するユーティリティ。それが、386|TOOLBOXです。
386|TOOLBOXは、386|DOS-Extenderというユーティリティを使用して、32bitプロテクトモードという従来のMS-DOSが十分に活用していなかった80386/486CPUの特長を、MS-DOSから100%活用可能にしました。
386|TOOLBOXは米国Phar Lap Software社の開発した386|ASM, 386|LINK, 386|LIB, RUN386, MINIBUG, CFIG386で構成されているユーティリティです。これらはアセンブラ、リンカ、ライブラリアン、386|DOS-Extender、デバッガ、コマンドオプションのカスタマイズプログラムであり、プロテクトモードプログラムの開発環境を提供します。

動作環境

必要なハードウェア　対応機種
NEC 32bit機対応版
- NEC　PC-9800シリーズ (XL2, RA, RL, LS, ES, RA21/51, RS21/51, T, DA, DS, NS)
- NEC　PC-H98シリーズ(Model 60, 70, 100)
- EPSON　PC-386シリーズ(386, LS, V, M, G)

IBM PC/AT互換機対応版
- 東芝　J-3100(SGT, SGX, GXS), J-3300
- IBM　PC/AT互換機, 各種AXマシン(80386/SX搭載機)

富士通 32bit機対応版
- 富士通　FMR-50(HV, S2, TX), FMR-70(HX1, HX2, HX3)

ハードディスク　1MByte以上のプロテクトモードメモリ
必要なソフトウェア　MS-DOS

NEC, EPSON機	MS-DOS	ver3.3以上
IBM PC/AT機, AX機	PC-DOS	ver3.2以上
東芝 J-3100等	日本語DOS	ver3.1
	英語DOS	ver3.3
富士通 FMR	MS-DOS	ver3.1 Level28以上

定価 ¥148,000

※すべての価格に消費税は含まれておりません。

NDP FORTRAN386

NDP FORTRAN386は、FORTRAN77仕様に完全準拠した80386CPUプロテクトモード専用FORTRANコンパイラです。
最大の特長である、プロテクトモードの4GByteのアドレス空間と、最適化による高速コードの生成により、VAX8600/SUN3等の汎用コンピュータ／ワークステーションクラスの規模と速度を持つ、アプリケーションの開発を可能にしました。使用の際は、386|TOOLBOX(弊社取り扱い製品)が必要です。

動作環境

386|TOOLBOXの動作環境に依存(ただし、富士通版はありません)。
プロテクトモードメモリ　2MByte以上
数値演算コプロセッサ

定価 ¥148,000

SPACE-II

科学技術演算ライブラリSPACE-IIは、学会論文・雑誌論文などに発表された最新のアルゴリズムを採用して作られた、数値演算に必要となるサブルーチンの汎用科学技術演算ライブラリ集です。
使用の際は、NDP FORTRAN386(弊社取り扱い製品)が必要です。

定価 ¥98,000

386|DEBUG

386|DEBUGは、80386/486CPUのためのシンボリックデバッガです。386|DOS-Extenderを利用する32bitプロテクトモードのプログラムのみならず、通常のMS-DOSのリアルモードプログラムをもサポートします。
使用の際は、386|TOOLBOX(弊社取り扱い製品)が必要です。

定価 ¥58,000

386|VMM

386|VMMは、386|TOOLBOXに含まれる386|DOS-Extenderに、ハードディスクを利用した仮想記憶機能を付加する仮想記憶マネージャです。
使用の際は、386|TOOLBOX(弊社取り扱い製品)が必要です。

定価 ¥86,000

パックセット発売

2点パックセット

386|VMM＋386|DEBUG	¥129,600
386|TOOLBOX＋386|DEBUG	¥175,100
386|TOOLBOX＋386|VMM	¥198,900

3点パックセット

386|TOOLBOX＋386|VMM＋386|DEBUG	¥248,200

TSS テックソフト&サービス株式会社 コンピュータ事業部
〒273 千葉県船橋市本町7-11-5 レランドセンタービル5F
Tel:0474-23-1322(営業直通) Fax:0474-23-0202
Tel:0474-22-9355(ユーザーサービス直通 10:00〜12:00, 13:00〜16:00)

〈資料請求番号001〉

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

V.3.0 いよいよ登場。
プログラミングエディター新世紀。

よりパワフルで、よりエレガントな
プログラミングのために……。

●C言語で組めるマクロ 新機能
使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能
ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉新機能
C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ
各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集
Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能
キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン(オプション)、メニュー(オプション)etc。

アメリカでも日本でも高い評価
●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
●まるで夢のようなアンドゥだ――おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能
◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円
(消費税は含まず)

■5機種対応パッケージ
(東芝、NEC、AX協議会、IBM、富士通)
●5.2DD、3.5 2DDデュアルメディア●MS-DOS3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

〈資料請求番号002〉

IBM Authorized Dealer

DOS/V OADG のサポート拠点です！

世界のソフトウェアが走る。

T·ZONE

仕事も遊びもちょっと、こだわる。そこで…

PS PLAZA

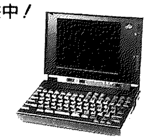
T·ZONEは今年1月19日からDOS/Vのサポート拠点として「PS PLAZA」をオープンしております。

特徴その1　日本IBMのSEが親切にどんなご相談にもお答えします。
特徴その2　いろいろな互換機上でDOS/Vが動くマルチプラットフォーム環境が揃っています。
特徴その3　PS PLAZAの直通番号があります。(03)3257-2654
特徴その4　外人さん大歓迎！英語、韓国語、ポルトガル語、イタリー語、スペイン語、バングラ語などのマルチリンガルOKです。
特徴その5　AT環境のボード、ソフトが日本一！揃っています。
特徴その6　DOS/V、WINDOWS、LAN、マルチメディアに関する、ホットな最新情報が得られます。

NOTE　ノートといえば、PS/55 note 大特価提供中！

IBM	PS/5523SJ1	~~¥248,000~~
IBM	PS/5523SJ4	~~¥368,000~~
NBCC	386NOTE	¥298,000
NBCC	386LAPTOP	¥266,000
NBCC	286NOTE	¥244,000
NBCC	286LAPTOP	¥220,000

PS/55 note



T·ZONE 386/33MHz NOTE RAM 2MB/max14MB. 3.5″FDD & 40MB HDD. 32階調VGA、外部CRTコネクター　¥498,000

DOS/V　DOS/VといえばT·ZONEです。入門機から互換機まで！

IBM	PS/5530-SW4	大特価提供中！ ~~¥609,000~~
	標準で6MBのRAMと日本語WINDOWS 3.0が付属しています。勿論DOS/Vも付いています。	
IBM	PS/5510ZJ2	大特価提供中！ ~~¥214,000~~
IBM	PS/5510ZJ4	大特価提供中！ ~~¥308,000~~
IBM	PS/5510TJ4	大特価提供中！ ~~¥564,000~~
MITAC	386SX/16.RAM 2MB.3.5″FDD.40MB HDD.モニター	¥298,000
MITAC	386SX/16.RAM 3MB.3.5″FDD.40MB HDD.モニター	¥318,000
MITAC	386SX/16.RAM 5MB.3.5″FDD.40MB HDD.モニター	¥338,000
MITAC	386SX/16.RAM 8MB.3.5″FDD.40MB HDD.モニター	¥378,000
MITAC	386SX/16.RAM 2MB.3.5″FDD.100MB HDD.モニター	¥348,000
MITAC	386SX/16.RAM 5MB.3.5″FDD.100MB HDD.モニター	¥388,000
MITAC	386SX/16.RAM 8MB.3.5″FDD.100MB HDD.モニター	¥428,000
MITAC	386SX/20.RAM 5MB.3.5″FDD.40MB HDD.モニター	¥368,000
ALR	POWER FLEX 20CSX. RAM 5MB. 3.5″ 85″. 40MBHD	¥385,000
ALR	BUSINESS VEISA 486/33. RAM. 2MB. 3.5″ 85″. 120MBHD	~~¥1,250,000~~
東芝	DynabookでDOS/Vが動いています(2Fでデモ中)	

VGA　DOS/Vの必須アイテムはVGAアダプターです。豊富な在庫から選べます。

BASIC VGA 640×480	¥19,800	SUPER-VGA 1024×768 256色	¥44,800
SUPER-VGA 1024×768 16色	¥38,000	Dyanbook用VGAアダプタ	~~¥40,800~~

VGAモニター　DOS/V対応やWINDOWS用として高解像大画面のモニターがほしいあなた、何を選びますか？

VORTEC	CM-1413	14″0.28mm 15.0-38kHz 1024×768	¥68,000
三菱	XC-98V3	14″0.28mm 98/VGA 640×460	¥78,000
ACER	7015	14″0.31mm 15.0-36kHz 800×600	¥85,000
日立	CM1483MU	14″0.28mm 30-40kHz 1024×768	¥89,000
NEC	MULTISYNC II	14″0.31mm 15.5-35kHz 800×560	¥95,000
NEC	MULTISYNC 3D	14″0.28mm 15.7-38kHz 1024×768	¥180,000
NEC	MULTISYNC 4D	16″0.28mm 30.0-57kHz 1024×768	¥280,000
IDEK	MF-5117	17″0.28mm 20.0-50kHz 1024×768	¥320,000
IDEK	MF-5121	21″0.31mm 21.0-50kHz 1024×768	¥480,000
IDEK	MF-5321	21″0.31mm 30.0-80kHz 1260×1280	¥580,000
IBM	8515	14″0.28mm VGA. 8514/A. XGA	~~¥160,000~~

WINDOWS 3.0　時代のトレンドWINDOWS 3.0がIBMではマルチウィンドウズで複数のDOSアプリが同時実行可能です。

IBM　日本語WINDOWS 3.0	¥20,000
ADDBE TYPE MANAGER	¥20,000
AFTER DARK	¥9,800
BECKERTOOLS	¥16,000
MICROSOFT ENTERTAINMENT PACK	¥8,800

その他WINDOWS用アプリケーションが200種類以上あります。
WINDOWS 3.0対応のPDS(英語)もあります。

大容量記憶装置　WINDOWSには大容量の記憶装置が必要です！

MINI-SCSIでFDタイプのNOTEなどに外部大容量HDDを接続できます。(プリンタポート接続)J-3100 AXなどもOK！

40MBセット	¥83,000	87MBセット	¥138,000	MINI-SCSI	¥29,800

1GBの大容量ハイスピード(35ms)の光ディスクがお薦めです。

AT用セット	¥570,000	三洋　CD-ROMセット(AT用)	¥128,000
MCA用セット	¥580,000	SCSI I/F 8 BIT	¥15,000

メモリー　PS/55用のメモリーがお買い得です!!

PS/5530用 2MB　¥32,300　PS/5530用 4MB　¥58,000　PS/5523用 4MB　¥58,000

マルチメディア

フルモーションビデオやHiFiオーディオ、イメージキャプチャー、圧縮などの先進のテクノロジーがT·ZONEにいっぱいです。

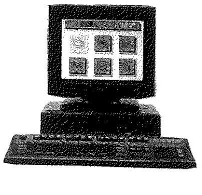
M-MOTIONデジタルビデオをデモ中！
VGAでテレビがみれます!!!
WINDOWSでフルカラーのイメージの取り込みがOK！

GAME　時代はゲーム!? TERADRIVEや新作のソフトが山盛りでご来店をお待ちいたしております。

SEGA TERADRIVE MODEL1	~~¥148,000~~	大特価提供中！
SEGA TERADRIVE MODEL2	~~¥188,000~~	大特価提供中！
SEGA TERADRIVE MODEL3	~~¥248,000~~	大特価提供中！
SEGA TERADRIVE 専用モニター	~~¥89,000~~	大特価提供中！
SEGA TERADRIVE マウス	~~¥6,800~~	大特価提供中！

WALT DISNEYの永遠のアイドル、ミッキーマウスがあなたにしゃべりかけます。

MICKEY'S ABC	¥10,800
MICKEY'S 123	¥10,800
MICKEY'S ABC & SOUND SOURCE	¥13,800
SOUND SOURCE(驚異のローコスト音源スピーカーです)	¥6,800
ADLIB FOR ATバス	¥30,000
ADLIB FOR MCAバス	¥40,000
SOUND BLASTER FOR ATバス	¥48,000
SOUND BLASTER FOR MCAバス	¥68,000
JOYSTICK	¥4,800
ゲームカード(PC/AT)	¥9,800
ゲームカード(MCA)	¥15,000

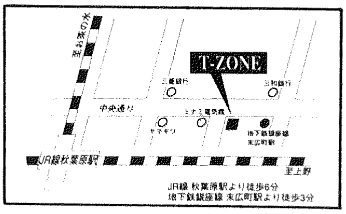


お知らせ　DOS/Vプログラムコンテスト開催。～9月末

DOS V用のユーティリティーやFEP、表計算ソフト、エディターなど、どんなジャンルのソフトでもOKです。
PDS、フリーウェア、シェアウェアをどしどし作って下さい。
DOS V用であれば何でもOK！

特選1名様に「テラドライブ」をプレゼント……他、お申し込みの方全員にもれなく商品をさしあげます。ふるってご応募下さい。

T·ZONE
IBMゾーン
☎03-3257-2652



〒101 千代田区外神田4-4-1
☎03-3257-2652 FAX 03-3257-2661

T·ZONEスタッフ募集　IBM、Apple、無線のスタッフを募集中です。アルバイトも募集中！　☎03-3257-2690

<資料請求番号003>

米国「The C Users Journal」「TECH Specialist」提携誌

C JOURNAL 9

SEP.1991

C言語を主体とするコンピュータプログラマのための月刊情報誌

月刊 Cジャーナル

POLYWELL 80486が当たる!!
3カ月連続
新創刊記念
プレゼント
第3弾

新創刊3号 8月8日発売

定価980円(本体951円)

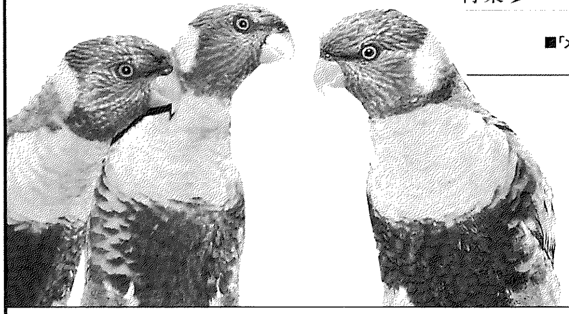
特集 メモリマネジメント

■DOSのメモリ管理の手法
■「メモリが不足です」救いの神はメモリ・マネジャー
■メモリアロケーションのデバッグ

第二特集
C言語を使う

付録DISK(5"2HD)
■fdpc, fd55, fd98
■simv
■simkey
■tr286
■CUGPDS #224 Utilix
■掲載記事の全コード収録

●連載●
★STANDARD C
★Dr.C's Pointers
★Questions & Answers
★たのむぜ!DOS/V!
★GUI環境
★Programming Style



株式会社ビレッジセンター 出版局
〒101 東京都千代田区三崎町3—1—16 神田アメレックスビル2F
●営業部:03-3221-1825●編集部:03-3239-1950

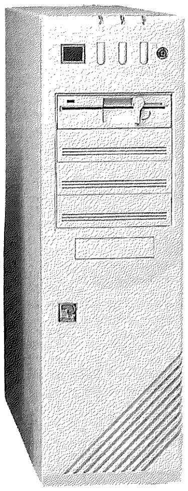
C JOURNALは毎月8日全国の書店でお求めになれます。

Polywell 80486DX Power IBM PC Compatible System

80486DX-25 or 33
AMI BIOS
128K Cashe
RAM Up to 32MB on Motherboard
Full Size Tower
220 Watt Power Supply
Western Degital IDE HD/FD Controller
Maxter HD(200MB)
TEAC 3.5"FDD*1
TEAC 5"FDD*1
2Serial/1Parallel I/F
Orchid ProII Super VGA Card(512K)

Keytoronic 101 Keyboard
MicroSoft Mouse
MS-DOS 4.1
MS-Windows 3.0

HITACHI 14" Super VGA Display
(1024*768 non-interlaced)
初期不良無償交換。本体1年間保証。



25Mhz		40MB HD	80MB HD	120MB HD	200MB HD
	1MB RAM	¥559,000	¥584,000	¥600,000	¥656,000
	2MB RAM	¥573,000	¥598,000	¥614,000	¥670,000
	4MB RAM	¥589,000	¥614,000	¥630,000	¥686,000

33Mhz		40MB HD	80MB HD	120MB HD	200MB HD
	1MB RAM	¥600,000	¥625,000	¥641,000	¥697,000
	2MB RAM	¥614,000	¥639,000	¥655,000	¥711,000
	4MB RAM	¥630,000	¥655,000	¥671,000	¥727,000

米国The C Users Group日本支部
VILLAGE CENTER INC.

■本部:〒101 東京都千代田区三崎町3-1-16神田アメレックスビル2F
TEL.03-3239-2187 FAX.03-3221-1768
■Shop:〒101 東京都千代田区神田神保町1-35-15 だるまビル1F
TEL.03-3219-6113

スタッフ募集中!!▶明るくヤル気のある方大歓迎です。詳しくは電話3239-2187/担当:中村までお問い合わせください。

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集
グラフィックスプログラミングの世界
基礎からレイトレーシングまで，プログラミング技法を徹底解説

グラフィックスといってもいろいろあるけれど…… 2
グラフィックスプログラミング入門 小林龍生

1677万色・フルカラーグラフィックスプログラミング 14
フレームバッファ活用法 尾崎行雄

超リアルな画像作成テクニック 26
レイトレーシング入門 稲葉正彦

DOS/V で注目される世界標準機のグラフィックス 36
IBM PC のビデオシステム入門 柳澤健二

ハードウェアに依存しないグラフィックス 47
Windows におけるグラフィックスプログラミングの実際 中新田秀美

市販フレームバッファ・グラフィックスライブラリ製品紹介 63

パソコン博物館 第9回 66
インテリジェントペリフェラルコントローラ／日立 ベーシックマスターレベル3
井戸禎光

情報処理技術者試験 第2種合格ノトレーニング 113
第6回 商業 佐々木秀夫

表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

プログラマの道具学　第9回
シンセサイザ　祐安重夫 …… 68
Book Review「続ソフトウェア千夜一夜物語」　中村正三郎 …… 118
本のアンテナ　古田島義和
佐々木哲のニューヨークシステム通信 …… 120
西川和久の希望的観測 …… 122
コンピュータの法律　熊谷信太郎 …… 124
アキバ短信
プログラマの仕事場第6回　アンテナハウス株式会社 …… 126

プログラム「遊」工房　九乃巻
「射撃」ゲーム　谷口典生 …… 70
IC 千夜一夜　第3回
8ビットマイクロプロセッサの発展ー8008と8080ー　虎岩登 …… 76
MS-DOS ツール・ラボ　第6回
コンパイラの自己記述　FORTHコンパイラ②FAST　加野島英渡 …… 80
実践C塾　基礎テクニック篇　第3回
文字列の入力（後編）　大野元久 …… 91
98活用アセンブラ技法／番外編　第1回
アセンブラ命令コードの把握　星野操 …… 103

INFORMATION PAGE …… 129
NEWS & NEW PRODUCTS …… 130
Programmer's Program Final Super Pack …… 134
オンラインソフト採集帳9「EXEXE.EXE」 …… 136
HEALTHY? UNHEALTHY?
ドラゴン・ポスト …… 138
付録ディスクのご案内 …… 140
バーチャルスペースガイド …… 141
読者プレゼント
クイズ・ランダムスクリーン …… 142
オンラインサービス（SE-NET）のお知らせ
バックナンバーのご案内 …… 143

付録
5"2HD フロッピーディスク
綴じ込み付録
PC-9801シリーズ・グラフィック BIOS/LIO 機能一覧

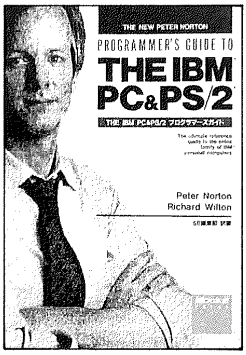
広告索引　　資料請求番号
AX育樹会 …… P02
テックソフト＆サービス(株) …… 001
日本システム(株) …… 002
T・ZONE …… 003
(株)ビレッジセンター …… 004
丸文(株) …… P03
三菱電機(株) …… P04

人材募集広告
(株)数理技研
(株)エルテックス
(掲載順)

1991 SEPTEMBER No.9

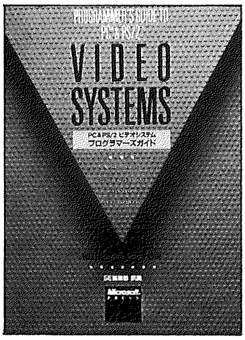
プログラミングのための優良図書

IBM PC & PS/2 プログラミング3部作



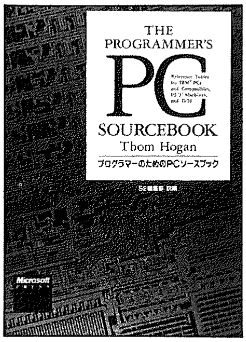
THE IBM PC & PS／2 プログラマーズガイド

●Peter Norton & Richard Wilton著
●定価5,200円
あのピーター・ノートンのベストセラー。IBMプログラマー必携のスタンダードガイダンス。



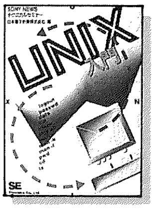
PC & PS／2 ビデオシステム プログラマーズガイド

●Richard Wilton著
●定価6,000円
IBM PCファミリーのビデオシステム完全ガイド。



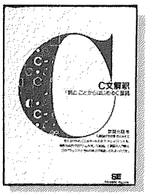
プログラマーのためのPCソースブック

●Thom Hogan著
●定価5,200円
IBM PC&PS／2プログラマ必携の技術資料完訳。



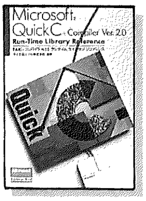
UNIX入門

●日本電子計算株編
●定価1,800円
SONY NEWSのUNIX講座に基づくUNIXの入門書。



C文解釈

●豊国永健著
●定価2,000円
C言語の文法書ではあまり取り上げられてこなかったわかりづらいポイントを、実際に応用プログラムを用いて解説。



Quick C コンパイラVer.2.0 ランタイムライブラリリファレンス
Quick BASIC Ver.4.5 ステートメント＆関数リファレンス

●マイクロソフト株式会社 監修
●定価各2,400円

マクロアセンブラ プログラマーズテキスト

●吉川敏則著
●定価2,600円

最低限必要なハードとソフトの知識、データ演算の仕組みを踏まえた上でアセンブラプログラミングの基本を丁寧に解説。

マイクロソフトの超人気言語ソフトQuickシリーズ。Quick BASIC Ver.4.5、Quick C Ver.2.0スタンダードリファレンス。マイクロソフト社監修でオンラインヘルプを完全書籍化。サンプルプログラム満載のリファレンス。

SE Shoeisha Co., Ltd. 株式会社翔泳社 〒102東京都千代田区平河町2-4-14平河町KSビル TEL.03-3263-0447(代表)

翔泳社の本は全国どの書店でもお求めになれます。店頭にない場合は「至急」とお申し出の上ご注文いただければ10日位でお届けできます。

〈資料請求番号005〉

GRAPHICS PROGRAMMING

特集
グラフィックス プログラミングの世界

基礎からレイトレーシングまで，プログラミング技法を徹底解説

数年前まで，本格的なコンピュータグラフィックスは，高価なワークステーションを用いなければできないものと考えられていた．しかし，パーソナルコンピュータのハードウェア性能の向上，個人にも手のとどく手ごろな価格のフルカラーフレームバッファが発売されたことなどによって，パーソナルコンピュータで十分実用的なコンピュータグラフィックスが可能になってきている．
本特集では，グラフィックスプログラミングの基礎として，ディスプレイ画面上に1つの点を打つ，1本の線を引くといった，基本の基本を解説し，次に1677万色フルカラーの世界を実現するフレームバッファを利用したプログラミングテクニック，およびリアルな3次元画像を実現するレイトレーシングの技法を紹介する．さらに，DOS/Vの登場によって注目を集めているIBM PCのグラフィックスモードとデバイス独立を実現したMS-Windowsのグラフィックスを解説する．

GRAPHICS PROGRAMMING

特集

グラフィックス
プログラミング
の世界

グラフィックスといってもいろいろあるけれど…

グラフィックスプログラミング入門

小林龍生

お絵描きソフトと呼ばれるペイントツールからレイトレーシング，アニメーションまで，現在ではさまざまなかたちでコンピュータグラフィックスが実現されている．
しかし，これらの高度なグラフィックスプログラムを実現する基本は，コンピュータの画面上にたった1つの点を打つことにある．点を打つことさえできれば，すべてのグラフィックスルーチンの記述が可能だといえる．グラフィックスプログラミングの入門として，Turbo Pascal 6.0を用いた点の打ち方から，グラフィックスを実現するアルゴリズムを理解することの重要性について解説する．

プログラムがやっていること

たとえばこんなプログラムを考えてみよう．サイコロゲームのオープニング．2個のサイコロが投げられて，回転しながらUFOのように自在に画面を飛び回る．

これをPC-9801で表現しようというわけ．バックにはリアルなイラスト．光が僕たちからみて左後方上部から当たっている．途中で2個のサイコロが交差する場面もある．実のところ，こんなのをまじめに計算していたら，現在のPC-9801では80386のCPUを使ったって実用的な速度は出やしない．この際，そんなことは無視して，やらなければならないことを考えてみることにしよう．

いちばんまじめというか短絡的というか力ずくの方法は，本格的なモデリング言語で記述して，レイトレーシングを用いてバックまで処理してしまうこと．CGプロダクションで日夜行われているニュースショーのオープニングタイトルなんかはこの類だ．これなら，プログラムの知識がなくても何とかなる．どこかのデザイン専門学校のCGデザイン科とかに入れば，作り方は詳しく教えてもらえるだろう．しかし，僕に求められていることはもうちょっと違うことだと思う．もう少し手作りというか手抜きというか自分で工夫して適当なプログラムをでっち上げようというわけだ．

実のところ，以前これに似たことをあるメーカーの店頭デモ用にやったことがある．実際には，リアルタイムに計算したのではまにあわないので，前もってコマ割りごとに計算して準備しておいたものを，ブロック転送で処理するようにした．

そのときは，Zバッファ法を用いた簡単なモデリング，レンダリングシステムを自分で作って利用した．このシステムは，トリスと名づけられて，しばらくはBBSの仲間たちが使って遊んでいたが，いつのまにか熱も冷めてしまった．

このトリスにしたところで，一朝一夕にできあがったわけではないけれど，一夜漬けでこれを作ることを想定して，どれだけのことが必要かを駆け足で見ていくことにしよう．ま，僕がたどってきたグラフィックスについての勉強の過程を一気に復習しようというわけ．

さしあたり，ハードウェアをNECのPC-9801シリーズのアナログ版に，処理系をボーランドのターボパスカルに限定する．別に他意があるわけではなく，僕の常用のシステムがそうなっているというだけのこと．Cのユーザーにひとこといい訳をしておくと，グラフィックスをやるならパスカルを読めるようにだけはしておいたほうがいいと思う．将来グラフィックスに関して，ある程度つっこんで文献を調べようとしたら，結構な頻度でパスカルまたは疑似パスカルのアルゴリズム記述言語にぶつかることになるから．

点を打つことに始まり……

僕の友人でサピエンスのスーパータブローの開発者としても知られる安斎利洋にいわせると，グラフィックスは点を打つことに始まり点を打つことに終わるということになる．点が打てなければテンで話にならないのである．点が打てれば，原理的にはグラフィックスのルーチンはすべて高級言語で記述することができる．

グラフィックスを大きく分けると，線で表現するものと面で表現するものの2つになる．以前よく目にしたのが，ワイアーフレームという針金細工みたいに直線だけで表されたCG．CADなんかではこういったのがまだ多い．一方，面で表現するものとしては，いわゆるサーフェスモデルなんかがこれに相当する．レイトレーシングというのは，意外に思うかもしれないが点を基礎とした描画方法だ．ペイントツールというのも原理的にはCRT上に点を1つ1つ置いていく形式になっている．

点を打つという儀式

コンピュータというのは，外からの何らかの刺激に対してある種の操作を施して反応を返すシステムと考えられる．使う人間の側からいくと，自分とシステムとの間に，ある種のコミュニ

リスト1　Pset手続き

```
procedure Pset(x,y,color:integer);
```

リスト2　Turbo Pascal 6.0マニュアルのPutPixel手続き

```
procedure PutPixel(x,y:integer; Pixel:word);
```

ケーションが成り立つことが前提になっている.

たとえば電話をかけるときの「モシモシ」とか，マイクをテストするときの「テスッテスッ」とか，入力に対して出力のある系では，必ずといっていいほど系の回路がつながっているかどうかを調べる儀式が存在する.

CRTに点を打つというのは，そんな儀式の1つと思えばいいかもしれない．僕は新しいハードなり処理系を手に入れると，なによりもまず点を打つルーチンを使ってCRT上のある区間を塗りつぶすプログラムを書いてみる．そう，カーニハン&リッチーが『プログラミング言語C』ではやらせた“Hello World!”に相当すると思えばいいだろう.

つまり，リスト1の手続きを何とか実現すればよい.

現在，ターボパスカルはVer.6.0になっていて，グラフィックスに関してもBGI (Borland Graphic Interface) というCやC++と共通で使える非常に充実したライブラリをもっている．マニュアルを見ると，リスト2のような手続きがしっかり載っている．処理系に付属してくるグラフィックスライブラリを使いたい方はどうぞ．ここで教訓.

あるものは利用しろ．自分が使う処理系のマニュアルは徹底的に調べろ.

ターボパスカルによるPset手続き

しかし，僕がPC-9801を使いはじめたころのターボパスカルは，Ver.3，MS-DOS汎用の処理系で，どんなMS-DOSマシンにも簡単にインストールして利用することができたかわりに，ハードウェアにディペンドした機能は皆無だった．もちろん，グラフィックスのルーチンが付属しているわけはない．ここで最初の突破口となったのが，Pset手続きを作ることだった.

PC-9801は，アドレス空間上にVramをマッピングしている．このアドレスにターボパスカルがもっている絶対アドレスにアクセスする機能を利用してアクセスすれば，とくにアセンブラを使用しなくても手続きを書くことができる．しかし，PC-9801のVramは，いわゆるプレーン型の構造になっており，Vramのあるアドレスにアクセスすると，横に8個のピクセルに影響をおよぼしてしまう．ピクセル1つだけを変化させたいとすれば，同一のアドレス内でマスキングをしなければならない．逆に広い面積を一気に塗りつぶすときなどは高速に処理ができる（リスト3参照).

このアセンブラを使わずに書いた点を打つ手続きをベースにして，僕たちはグラフィックスライブラリを構築していった．このライブラリが基となって『ターボ・グラフィックス』（安斎利洋，伊吹龍著，JICC出版局）という本ができあがった.

現在僕自身が使っているライブラリの中では，この点を打つ手続きはアセンブラに書き換えられて，ずいぶん高速になっている．しかし，それ以外の部分はすべてパスカルのまま．これは半分意地みたいなものなのだけれど，ハードウェアに依存する部分は，この点を打つ手続きだけに押し込んで，それ以外はポータビリティを高めておくというのがいいように思う.

PC-9801以外の機種やターボパスカル以外の処理系を使っている方は，自分でこれに相当するルーチンを書いてみるとよいだろう．実際には，既存のライブラリを使うとしても，自らの手でハードウェアに直接アクセスした経験は必ず役に立つはずだ．レーサーなんかでも一度は自分の走るレーシングコースを歩いてみるということだし.

線を引くための考え方

僕が作ったトリスは，すべての物体

グラフィックス用語集
アクセラレータ　グラフィックス描画は解像度や色数が増えるにしたがってCPUにかかる負担も多くなる．グラフィックスアクセラレータはこれらの処理をCPUに代わって行う.

リスト3　PSETETC.PAS

```
{ターボパスカルによる点と線と円}
const
  HorWidth = 640;
  VerWidth = 400;
  MaxHor   = HorWidth - 1;
  MaxVer   = VerWidth - 1;
  HorBytes = HorWidth div 8 - 1;
  OutOfFrame = $FF;
  {$IFDEF ANALOG8COL}
    MaxPlane = 2;
    MaxColor = 7;
  {$ELSE}
    MaxPlane = 3;
    MaxColor = 15;
  {$ENDIF}

type
  LusterT    = array[0..HorBytes] of byte;
  VramT      = array[0..MaxVer] of LusterT;
  RgbT       = (Red,Green,Blue);
  PaletteT   = array[0..15,Red..Blue] of byte;

const
  InitialPalette : PalettoT = (
     ( 0, 0, 0), ( 0, 0,15), (15, 0, 0), (15, 0,15),
     ( 0,15, 0), ( 0,15,15), (15,15,00), (15,15,15),
     ( 7, 7, 7), ( 0, 0,10), (10, 0, 0), (10, 0,10),
     ( 0,10, 0), ( 0,10,10), (10,10, 0), (10,10,10) );

const
  NormalExit: boolean = true;

var
  Vram0       : VramT absolute $A800:$0000;
  Vram1       : VramT absolute $B000:$0000;
  Vram2       : VramT absolute $B800:$0000;
  Vram3       : VramT absolute $E000:$0000;
  PaletteData : PaletteT;

procedure PSet( x,y: integer; color: byte );
var Xpos,dAddr,OnPattern,OffPattern: byte;
begin
  if (x > MaxHor) or (x < 0) or (y > MaxVer) or (y< 0) then exit;
  Xpos := x shr 3;
  dAddr := x and 7;
  OnPattern := $80 shr dAddr;
  OffPattern := not OnPattern;
  WaitGDC;
  if ( color and 1 ) = 0
    then Vram0[ y,Xpos ] := Vram0[ y,Xpos ] and OffPattern
    else Vram0[ y,Xpos ] := Vram0[ y,Xpos ] or OnPattern;
  if ( color and 2 ) = 0
    then Vram1[ y,Xpos ] := Vram1[ y,Xpos ] and OffPattern
    else Vram1[ y,Xpos ] := Vram1[ y,Xpos ] or OnPattern;
  if ( color and 4 ) = 0
    then Vram2[ y,Xpos ] := Vram2[ y,Xpos ] and OffPattern
    else Vram2[ y,Xpos ] := Vram2[ y,Xpos ] or OnPattern;
  {$IFNDEF ANALOG8COL}
  if ( color and 8 ) = 0
    then Vram3[ y,Xpos ] := Vram3[ y,Xpos ] and OffPattern
    else Vram3[ y,Xpos ] := Vram3[ y,Xpos ] or OnPattern;
  {$ENDIF}
end;{of Pset}

procedure Line( x1,y1,x2,y2: integer; color: byte );
var sx,sy,dx,dy,i,s,x,y: integer;
begin
  if x2>x1 then sx := 1
           else sx := -1;
  if y2>y1 then sy := 1
           else sy := -1;
  dx := abs( x2 - x1 );
  dy := abs( y2 - y1 );
  if dx > dy then
    begin
      s := dx shr 1;
      for i := 0 to dx do
        begin
          Pset( x1,y1,color );
          s := s + dy;
          if s > dx then
            begin
              s := s - dx;
              y1 := y1 + sy
            end;
          x1 := x1 + sx
        end
    end
  else
    begin
      s := dy shr 1;
      for i := 0 to dy do
        begin
          Pset( x1,y1,color );
          s := s + dx;
          if s > dy then
            begin
              s := s - dy;
              x1 := x1 + sx
            end;
          y1 := y1 + sy
        end
    end
end;

procedure Circle( xc,yc,r: integer; color: byte );
var xi,yi,delta,deltal,Limit: integer;
begin
  xi := 0;
  yi := R;
  deltal := (1-R) shl 1;
  Limit := 0;
  while yi>=Limit do
    begin
      Pset(xc+xi,yc+yi,color);
      Pset(xc+xi,yc-yi,color);
      Pset(xc-xi,yc+yi,color);
      Pset(xc-xi,yc-yi,color);
      if deltal<0 then
        begin
          delta := deltal shl 1 + yi shl 1 - 1;
          if delta<=0 then
            begin
              inc(xi);
              deltal := deltal + xi shl 1 + 1;
            end
          else {delta>0}
            begin
              inc(xi);
              dec(yi);;
              deltal := deltal + xi shl 1 - yi shl 1 + 2;
            end;
        end
      else if deltal>0 then
        begin
          delta := deltal shl 1 - xi shl 1 - 1;
          if delta<=0 then
            begin
              inc(xi);
              dec(yi);
              deltal := deltal + xi shl 1 - yi shl 1 + 2;
            end
          else {delta>0}
            begin
              dec(yi);
              deltal := deltal - yi shl 1 + 1;
            end
        end
      else {deltal=0}
        begin
          inc(xi);
          dec(yi);
          deltal := deltal + xi shl 1 - yi shl 1 + 2;
        end;
    end;
end;{of cirice}
```

をポリゴン（多角形）で表現する簡単なものだった．モデリング部分とレンダリング部分に分かれていて，モデリング部分には，ワイアフレームで全体の構成をチェックする機能が付いていた．この機能を表現するには直線がひけさえすればよい．『ターボ・グラフィックス』にもDDA（あとで説明する）で直線を引く手続きは入っているが，直線を引くためにどのような考え方があるかを，ちょっと説明しておこう．

直線の引き方

直線を引くには，2つの点をつないで，間に隙間が空かないように連続して点を打っていくことになる．高校生のころの数学で習ったことと思うが，直線は簡単な式で表現することができる．2点を (x_1, y_1)，(x_2, y_2) とすると，図1のようになる．

今僕は高校生用の数学の公式集を見てこの式をひき写したのだけれど，別にむずかしい式ではない．これをちょっと変形すれば図2のようになる．

これをプログラムに書けば，さしあたりは直線を引くことができる．ちょ

っと書いてみようか（リスト4）.

これで一応は線を引くことができる．しかし，問題は多い．たとえば僕が今アンチョコにしている公式集には前にあげた公式のあとに，

$$x_1 \neq x_2$$

という条件が付いている．この場合はｙの値にかかわらず，

$$x = x_1$$

ということになる．この条件を入れて上記の手続きを書き換えるとリスト5のようになる．

実は，これでも問題は大ありで，x1とx2，y1とy2の大小関係が考慮されていない．ま，ここらあたりは本誌の読者なら簡単に解決できるだろう．さらに，ループの中のｙを求める部分についても，毎回同一の演算を繰り返していて無駄が甚だしい．適当な変数を立てて無駄な演算を置き換えるだけで，ずいぶんプログラムは高速化できる．

しかし，もっと決定的なことが2点ある．1つは，処理に例外があるということ．もう1つは，この手続きが本質的に実数演算を前提としていること．この2点は，グラフィックスを考えていく上で避けて通れない重要な問題を提起している．

例外処理の問題

まずは，例外処理の問題から片づけよう．媒介変数という言葉を覚えているかな．今調べてみたけれども，これも高校用の公式集に載っておりますな．媒介変数ｔを使って，直線の式を書き直してみよう（図3）.

ここで，ｎは直線の分割数．プログラムに落とすとリスト6のようになる．

これで，例外の処理はすべてクリアできるし，2点の大小関係も問題なくクリアできる．たとえば，円を表現する場合やスプラインやベジエなどの自由曲線を表現する場合など，この媒介変数は非常に重要な役割を演ずることになる．

実数演算の問題

もう1つの問題は，実数演算．ご存知のように，コンピュータというのはバカだから整数演算は猛烈に得意なくせに実数演算となるとからっきしダメ．スーパーコンピュータが話題になるのは，普通のコンピュータがダメな実数演算を猛烈なスピードでこなすからにほかならない．だからこそ，いわゆる数値演算プロセッサがCPUとセットになって用意されていて，ダメなCPUと相互扶助をする仕組みになっている．しかし，世の中に数値演算プロセッサが標準装備されたパソコンなんてまだまだ少ないから，CPUの貧弱な演算能力でごまかすために実数演算を回避して整数演算で何とかする技法が発達している．

これが，いわゆるDDA（Digital Differential Analyzer:デジタル微分機）といわれる手法だ．先ほどの直線を書く手続きも整数演算だけで書くことができる．編集部は，プログラムの

リスト4　line手続き①

```
procedure line(x1,y1,x2,y2:integer);
var x,y:integer;

begin
  for x := x1 to x2 do
    begin
      y := round(y1+(y2-y1)*(x-x1)/(x2-x1);
      pset(x,y);
    end;
end;
```

リスト5　line手続き②

```
procedure line(x1,y1,x2,y2:integer);
var x,y:integer;

begin
  if x1<>x2 then
  for x := x1 to x2 do
    begin
      y := round(y1+(y2-y1)*(x-x1)/(x2-x1);
      pset(x,y);
    end else
  for y := y1 to y2 do
    pset(x1,y)
end;
```

図1　直線の式①

$$(x_2-x_1)*(y-y_1)=(x-x_1)\times(y_2-y_1)$$

図2　直線の式②

$$y=y_1+\frac{(y_2-y_1)\times(x-x_1)}{(x_2-x_1)}$$

グラフィックス用語集

アナログRGB　RGBの各色を1か0ではなく値を与え色を表現する方法．この方法によって1670万色程度の発色が可能になった．ただしメモリの関係から同時発色は16色程度．

ひき写しだけで原稿料を稼ぎやがってなどと文句をいうかもしれないけれど，読者に単行本を買わせるのも酷だから，『ターボ・グラフィックス』に用いたDDAによる直線を描く手続きを引用しておこう（リスト3参照）．名誉のためにはっきりさせておくけれど，この部分はこの本の共著者の安斎利洋が書いたもの．僕が書いたルーチンもあったのだけれど，8ビットのころにスピード競争をして，あえなく敗れ去ってしまった．

いずれにせよ，DDAに関する原理的な問題はブレゼンハムという学者がほぼすべて解決してしまっている．

実際のプログラミングの現場では，ほとんどの場合，根本的なアルゴリズムは明らかになっており，それをどうインプリメントするかが問題になる．ちょっとした工夫で全体のスループットを飛躍的に向上させることができたり，整数化の誤差による出力の乱れを抑え込んだりできる場合が少なくない．

要点の復習

さて，ここまでの復習．

グラフィックスというのは，抽象的な座標系に数式で表すことのできる図形を，適当な方法でCRT上の個々のピクセルに対応させるということ．

数式の扱いに例外を設けないためには，媒介変数を活用することが役に立つ．

実数演算を使わずに整数演算のみで図形を表現する方法としてDDAという考え方が役に立つ．

アルゴリズムを個々の処理系にインプリメントするときの技術や考え方の差によって，速度や精度に大きな差が現れる場合がある．

図3

直線の式③

$$x = \frac{x_2 - x_1}{n} \times t$$

$$y = \frac{y_2 - y_1}{n} \times t$$

（ただし n は分割数）

リスト6　line手続き③

```
procedure line(x1,x2,y1,y2:integer);
const step=適当な分割数
var
  t:integer;
  x,y:integer;
  deltaX,deltaY:real;
begin
  deltaX := (x2-x1)/step;
  deltaY := (y2-y1)/step;
  for t:= 0 to step do
    begin
      x := round(deltaX*t);
      y := round(deltaY*t);
      pset(x,y);
    end;
end;
```

リスト7　ポリゴンのデータ構造とDrawPolygon手続き

```
type
  PPoint = ^TPoint;
  TPoint = record
   Xcor,Ycor : real;
  end;


  TPolygon = PPoint;

procedure DrawPolygon(P:TPolygon);
var tLink : PPoint;
begin
  tLink := P;
  while tLink^.next<>nil do begin
    line(tLink^.Xcor,
         tLink^.Ycor,
         tLink^.next^.Xcor,
         tLink^.next^.Ycor);
    tLink := tLink^.next;
  end;
  line(tLink^.Xcor,
       tLink^.Ycor,
       P^.Xcor,
       P^.Ycor);
end;{Draw Polygon}
```

多角形から楕円まで

3つ以上の点を直線をつないでいけば多角形になる．あたりまえだ．このあたりで重要なのは，多角形のデータのもち方．一般的にはポインタを使って，リンクトリストでもたせるのが正解だと思う．たとえば，リスト7のようなものだ．

ポリゴンをリンクトリストで表現するためのデータ構造と，リンクトリストを走査して直線でつなぐ手続きをあげておく．これでわからない人は，申し訳ないけれどもアルゴリズムとデータ構造の教科書を見て欲しい．

グラフィックス用語集
アンチエリアシング　ジャギーなどの画像の不自然な部分を細部にぼかしを入れることにより防止する方法

データ構造の理解は必須

ドロー系の2次元グラフィックスにしても，3次元のグラフィックスにしても，データ構造の設計と扱いは必須のことになる．パスカルの設計者ニクラウス・ヴィルトが名著『アルゴリズム＋データ構造＝プログラム』（N.ヴィルト著，片山卓也訳，日本コンピュータ協会）の題名で端的に語っている真理は，グラフィックス関係のプログラミングでも例外ではない．むしろ，グラフィックスこそは，ヴィルトの名言を実践する最適の場だと思う．突き放していっちゃうと，リスト構造もろくにわからずにグラフィックスをやるなよな，ということになる．なんて書いちゃって，前述のアルゴリズムが違っていたらどうしよう．リスト構造の処理でバグを出すことって結構あるんだよね．

円の描画

円の描画についても，パラメトリカルな表現もDDAによる表現も可能だ．ブレゼンハムのアルゴリズムは，各象現に分けて4分の1ずつの円弧を発生させるようになっている．これを4回繰り返す奴はバカで，円の対象性に注目して手を抜くのが正解（リスト3参照）．

楕円の描画

x軸，y軸に関して対象性をもっている楕円に関しては，比較的簡単にDDA化が可能だけれど，回転してしまうとかなりやっかいになる．残念ながら，僕自身は楕円のDDA化はインプリメントしたことがない．パラメトリカルな表現を用いて適当な間隔で点列を発生させて，間を直線でつないでしまう．このとき，三角形の加法定理（図4）をうまく使うと，いちいち三角関数の値を計算しなくてすむ．

楕円の式を図5のようにしてインプリメントしてみよう．と，思ったけれどめんどうなのでこれは宿題ということにしておこう．

ポリゴンの塗りつぶし

トリスのレンダリング部分は，Zバッファアルゴリズムで陰面処理を施しながら，それぞれのポリゴンを走査していき，ポリゴン内のピクセルすべてについて明るさを計算するという方法を取った．このとき基礎になったのが，いわゆるポリゴンの塗りつぶしだ．

これが結構大物．点と線の世界と面の世界をつなぐのが，この手続きのもっとも大切な役割になる．応用範囲もとても広い．今あげた3次元グラフィックスのレンダリングにも使うし，最近はやりのアウトラインフォントなども最終的にはこの塗りつぶしの作業が不可欠になる．

具体的にはバケットソートという方法を使って，ポリゴンをバラバラの直線に分けてしまい，その直線とx軸に平行なスキャンラインの交点を求めることによって塗りつぶしを行うわけ．なんて書いてもわかるわけないよね．このアルゴリズムは，境界条件の判断でバグが出やすい．無理に自分でイン

図4
三角形の加法定理

$$\cos(\theta+d\theta)=\cos\theta\cos d\theta-\sin\theta\sin d\theta$$
$$\sin(\theta+d\theta)=\cos\theta\sin d\theta+\cos d\theta\sin\theta$$

図5
楕円の式

$$x=a\cos\theta$$
$$y=b\sin\theta$$

グラフィックス用語集
インターレース　ディスプレイの走査線スキャニングを偶数フィールドと奇数フィールドに分け，交互に描画する方法

プリメントしないで，身近にあるインプリメントを適当に変形して活用すればいい．一応僕がインプリメントした例をあげておく（リスト8）．

これにも実はネタがあって，以前大野義雄氏がピクセルに連載されていた記事と，David F.Rogersの『実践コンピュータグラフィックス』(David F. Rogers著，山口富士夫監修，日刊工業新聞社)．この2つからいいとこ取りで作ってある．僕がバグで苦労したのは，whileループの脱出条件で，パスカルの処理系が条件判断のためにポインタにnilが入っているところで，値を参照に行ってしまってハングするというものだ．現在のターボパスカルは，そのあたりの条件判断もかなり自由に設定できるオプションがあるのでずいぶん楽になっている．ここでもう1つの教訓．

処理系のもつ特質について理解しておこう．プログラムの安全性を高められる．

ここにあげたリスト8は，以前友人のプログラムに組み込むために作成したものなので，引数の渡し方が配列になっている．リンクトリストで渡すほうがずっとエレガントなのだけれど，ヒープの管理が若干やばいので，配列で渡すことにしてある．同様な理由で，手続きから抜けるときに，手続き内で使用したヒープ領域はきれいに掃除して抜けるようにしてある．

もうひとつ特徴をあげると，リスト構造の最後にいわゆる番兵を置くことにより，単純な境界条件だけで，走り抜けによるバグの発生を防いでいる．ここいらあたりは，データ構造とアルゴリズムの問題．

座標変換と行列演算

トリスのモデリング部分の主要な役割は，テキストファイルで与えられたポリゴンを階層化して，その階層ごとに回転，移動，拡大縮小の処理を施すことにあった．

サイコロの例で，移動してとか回転してとかの言葉で表されていたことは，すべてこの座標変換に還元される．ま，めんどうだといえばめんどうだけれど，これもクリアしておかなければ先はない．原理的には高校程度の座標幾何学で十分理解できる内容のはず．

最初は，それぞれの座標軸にそって x，y，zの値をゴチョゴチョ計算して

リスト8 POLFIL.PAS

```
unit polfil;
(*88・10・29
基本的考え方。
ポリゴンの塗りつぶし。最も基本的なもの。
ハーフラインスキャンニングを使う。
基本的アルゴリズムとしては、アクティヴエッジリストを用いたスキャンコンヴァー
ジョンを採用する。
リスト内の走査にはＬｉｎｋＬｉｓｔを用いる。Ｌｉｓｔの終点には、番兵を置く。
*)

interface

uses
graph98;

type
  Tag = array[0..1] of integer;{個々の点の座標を表す配列}
const
  TagTop = 128;{ポリゴンの最大頂点数、変更可}
  Xmax = 639;
var
  TagTable : array[-1..TagTop] of Tag;
procedure PolygonFill(NoOfTags,color:integer{Plot:plotType});

implementation



procedure PolygonFill(NoOfTags,color:integer);
type
  EdgePointer = ^Edge;
  Edge = record
            Yend:integer;
            Xnow,deltaX : real;
            next:EdgePointer;
         end;{record}

  CrossPointPointer = ^CrossPoint;
  CrossPoint =
    record
      X : real;
      next : CrossPointPointer;
    end;{record}

var
  Ybucket : array[0..399] of EdgePointer;
  AELhead : EdgePointer;
  CPLhead : CrossPointPointer;
  SentinelEdge : EdgePointer;
  SentinelCP : CrossPointPointer;
  Dummy1,Dummy2 : Pointer;

  procedure Initialize;
  begin
    New(SentinelEdge);
    with SentinelEdge^ do
      begin
        Yend := 400;
        Xnow := -1;
        deltaX := 0;
        next := nil;
      end;
    New(SentinelCP);
    with SentinelCP^ do
      begin
        X := Xmax+1;
        next := nil;
      end;
      AELhead := SentinelEdge;
  end;{Initialize}

  procedure SetYbucket;
  var
    yc,i:integer;

    procedure SetEdgeToYbucket(T1,T2:Tag);
    var
      tLinkEdge : EdgePointer;
      procedure AddEdgeToYbucket(yc : integer; tLinkEdge : EdgePointer);
      begin
        tLinkEdge^.next := Ybucket[yc];
        Ybucket[yc] := tLinkEdge;
      end;{AddEdgeToYbucket}

    begin
      if T1[1]=T2[1] then exit;
      new(tLinkEdge);
      if T1[1]<T2[1] then
        begin
          with tLinkEdge^ do
            begin
              Yend := T2[1];
```

グラフィックス用語集

陰面消去／陰線消去　かくれた面や線を消去する処理．3次元の物体をディスプレイに表示したとき，座標変換するだけでは裏側も表示され物体の形がつかみにくい．

```
                deltaX := (T2[0]-T1[0])/(T2[1]-T1[1]);
                Xnow := T1[0]+deltaX/2;
              end;
            AddEdgeToYbucket(T1[1], tLinkEdge);
          end
        else {T1[1]>T2[1]}
          begin
            with tLinkEdge^ do
              begin
                Yend := T1[1];
                deltaX := (T1[0]-T2[0])/(T1[1]-T2[1]);
                Xnow := T2[0]+deltaX/2;
              end;
            AddEdgeToYbucket(T2[1], tLinkEdge);
          end;
    end; {SetEdge}

  begin
    for yc:=0 to 399 do
      Ybucket[yc] := SentinelEdge;
    TagTable[NoOfTags] := TagTable[0];
    for i:=0 to NoOfTags-1 do
      SetEdgeToYbucket(TagTable[i], TagTable[i+1]);
  end; {SetYbucket}

  procedure Scan;
  var
    yc : integer;

    procedure ScanOneLine(yc:integer);

      procedure SetActiveEdgeList;
      var
        tLinkEdge, ttLinkEdge, F, B : EdgePointer;
      begin
        if Ybucket[yc] = SentinelEdge then exit;
        F := AELhead;
        B := nil;
        while F<>SentinelEdge do
          begin
            B := F;
            F := F^.next;
          end; {while}
        while Ybucket[yc]<>SentinelEdge do
          begin
            ttLinkEdge := Ybucket[yc];
            Ybucket[yc] := Ybucket[yc]^.next;
            ttLinkEdge^.next := SentinelEdge;
            if B=nil
              then
                begin
                  AELhead := ttLinkEdge;
                  B := AELhead;
                end
              else
                begin
                  B^.next := ttLinkEdge;
                  B := ttLinkEdge;
                end;
          end; {while}
      end; {SetActiveEdgeList}

      procedure ResetActiveEdgeList;
      var
        tLinkEdge, F, B:EdgePointer;
      begin
        F := AELhead;
        B := nil;
        while F<>SentinelEdge do
          begin
            if F^.Yend=yc
              then
                begin
                  if B= nil
                    then AELhead := AELhead^.next
                    else B^.next := F^.next;
                  F := F^.next;
                end
              else
                begin
                  B := F;
                  F := F^.next;
                end;
          end; {while}
      end; {ResetActiveEdgeList}


      procedure SetCrossPointList;
      var
        tLinkEdge:EdgePointer;
        tLinkCP, F, B:CrossPointPointer;
      begin
            CPLhead := SentinelCP;
            tLinkEdge := AELhead;
            while tLinkEdge<>SentinelEdge do
              begin
                {Create CrossPoint}
                new(tLinkCP);
                {Set Value to CrossPoint}
                tLinkCP^.X := tLinkEdge^.Xnow;
                {Set CrossPoint to CrossPointList}
                  F := CPLhead;
                B := nil;
                while F^.X<tLinkCP^.X do
                      begin
                        B := F;
                        F := F^.next;
                      end; {while}
                tLinkCP^.next := F;
                if B=nil
                  then
                    CPLhead := tLinkCP
                  else
                    B^.next := tLinkCP;
                {Increase Data of ActiveEdgeList}
                with tLinkEdge^ do
                  begin
                    Xnow := Xnow+deltaX;
                  end; {with}
                tLinkEdge := tLinkEdge^.next;
              end; {while}
      end; {SetCrossPointList}


      procedure ScanCrossPointList;
      var
        tLinkCP : CrossPointPointer;
        xc : integer;


      begin {ScanCrossPointList}
        tLinkCP := CPLhead;
        while tLinkCP<>SentinelCP do
          begin
            Line(round(tLinkCP^.X), yc,
                   round(tLinkCP^.next^.X), yc, color);
            tLinkCP := tLinkCP^.next^.next;
          end; {while}
      end; {ScanCrossPointList}



    begin {ScanOneLine}
      SetActiveEdgeList;
      ResetActiveEdgeList;
      Mark(dummy2);
      SetCrossPointList;
      ScanCrossPointList;
      Release(dummy2);
    end; {ScanOneLine}

  begin
    for yc :=0 to 399 do
      ScanOneLine(yc);
  end; {Scan}


begin {Main of PolyFill}
  Mark(dummy1);
  Initialize;
  SetYbucket;
  Scan;
  Release(dummy1);
end; {PolygonFill}

end.
```

いくのがわかりやすいだろう．そのうち，行列（いわゆるマトリックス）を使うようになる．この違いについて簡単に説明しておいたほうがいいだろう．めんどうなので2次元でやるね．原理は3次元でもまったく同じ．

要素としては，回転，拡大縮小，移動の3つがある．

回転

リスト9のような手続きを考えてみよう．(xs,ys)は，元になる点の座標，(xd,yd)は，回転してできた点の座標．thは，回転する角度．

拡大縮小

これは，x軸とy軸に関してそれぞれ独立だから別に1つの手続きにする

グラフィックス用語集
LIO(Logical Input Output)　PC-9800シリーズのROM内にあるグラフィックス描画を行うためのサブルーチン．

リスト9　rotate手続き

```
procedure rotate(xs,ys : real; var xd,yd : real; th:real);

begin
  xd := xs*cos(th)-ys*sin(th);
  yd := xs*sin(th)+ys*cos(th);
end;
```

リスト10　Scale手続き

```
procedure Scale(xs,ys : real; var xd,yd : real; ratioX,ratioY : real);
ratioX,ratioYはそれぞれの座標軸方向の拡大率。
begin
  xd := xs*ratioX;
  yd := ys*ratioY;
end;
```

必要はないといえばない（リスト10）.

リスト11　Trans手続き

```
procedure Trans(xs,ys : real; var xd,yd : real; lengthX,lengthY : real);
begin
  xd := xs+lengthX;
  yd := ys+lengthY;
end;
```

移動

リスト11．以上のような手続きだけで，図形のあらゆる移動変形が表現できる．でも，これって猛烈に能率が悪いんだよね．

直線を回転させながら移動させるプログラムを示しておく（リスト12）.

リスト12　サンプルプログラム①

```
program sample1;
var

begin
  for t := 0 to tMax do begin
    Scale(x00,y00,x01,y01,deltaRatioX*t,deltaRatioY*t);
    Scale(x10,y10,x11,y11,deltaRatioX*t,deltaRatioY*t);
    Rotate(x01,y01,x02,x02,deltaR*t);
    Rotate(x11,y11,x12,x12,deltaR*t);
    Trans(x02,y02,x03,y03,deltaLx*t,deltaLy*t);
    Trans(x12,y12,x13,y13,deltaLx*t,deltaLy*t);
    Line(x03,y03,x13,y13);
  end;
end.
```

行列演算

今あげたような座標変換は，一般的に行列の演算で表すことができる．ここで，これについて議論するのはあまり得策ではないだろう．もちろん行列同士の演算と行列とベクトルの演算は理解しておいてもらいたいのだけれど，具体的な変換操作がどのような行列演算に相当するかは，これまたアンチョコに書いて……，おっと高校用の公式集には，２行２列の行列を用いた回転の例はあがっていたけれど，それ以外はありませんな．この程度のことになると，大学教養過程の話題かもしれない．ま，深入りはやめておこう．

２次元の座標変換は，移動，スケーリングまでも含めると，３行３列のマトリックスで表現することができる．インプリメント部分は別にして，手続きとそれを使って例の直線の移動を表すプログラムをあげておこう（リスト13）.

これからもわかるように，マトリックスを使うと，計算量が非常に少なくなる．とくに，多くの点に同じような変換を施す場合の効果は劇的なものがある．さらに，プログラム全体の可読性を高める効果も無視できない．

リスト13　サンプルプログラム②

```
program sample2;

type
  Vector3 = array[1..3] of real;
  Matrix33 = array[1..3] of Vertor3;

const
  UnitMat33 : Matrix33 ((1,0,0),
                        (0,1,0),
                        (0,0,1))

procedure rotate[Ms:Matrix33; var Md:Matrix33; Th:real);
procedure scaleX[Ms:Matrix33; var Md:Matrix33; S:real);
procedure scaleY[Ms:Matrix33; var Md:Matrix33; S:real);
procedure transX[Ms:Matrix33; var Md:Matrix33; L:real);
procedure transY[Ms:Matrix33; var Md:Matrix33; L:real);
procedure Operate(Vs:Vector3; M:Matrix33; var Vd:Vector3);
var
  V1s,V2s,V1d,V2d : Vector;
  Vd : Vector;
  M : Matrix33;
  t : integer;

for t := 0 to tMax do begin
  M := UnitMat33;
  rotate(M,M,deltaTh*t);
  scaleX(M,M,deltaSx*t);
  scaleY(M,M,deltaSy*t);
  transX(M,M,deltaLx*t);
  transY(M,M,deltaLy*t);
  Operate(V1s,M,V1d);
  Operate(V2s,M,V2d);
  Line(Vd[1],Vd[2]);
end;
```

グラフィックス用語集

解像度　グラフィックスの表示能力の程度を示す．PC-9800シリーズの場合，高解像度は横640×縦200ドットを，専用高解像度は横640×縦400ドットをさす．

図6
ベジエ曲線

$$x = x_0(1-t)^3 + 3x_1 t(1-t)^2 + 3x_2 t^2(1-t) + x_3 t^3$$

曲線補間アルゴリズム

いくつかの点を滑らかな曲線でつなぐのが，曲線補間アルゴリズム．実際には2点の位置と一次微係数から，これらの条件を満たす3次曲線を求めるものが多い．

代表的なのは，スプラインとベジエ曲線．ポストスクリプトで名を馳せたベジエをちょいと見ておきますか．

4点 x_0，x_1，x_2，x_3 を考える．t を0から1まで動かして，その間の x の変化を求める（図6）．

最初にあげたサイコロの例でサイコロをUFOのように自由に飛び回らせる場合，適当な位置をあらかじめ用意しておき，それを曲線補間のアルゴリズムでつないでいくことになる．つまり，アニメーションなどでの時間方向の曲線補間ということになる．

この曲線補間アルゴリズムは，ほかにもCADや文字の生成などに欠かすことができない．深入りすると本が1冊書けてしまうので，興味のある方は研究してみるといいかもしれない．

3次元のポイント

僕自身はトリスというサーフェスモデルのプロトタイピングをやってみたことがあるけれど，正直いって3次元にはあまり興味がない．あまりにも数学で割り切れすぎてしまうのだ．それよりも，2次元でいい加減な要素を残しながらプログラムを作っていたほうがおもしろいものができそうな気がする．

枚数がたりなくなりそうなので，ここでは重要なポイントだけをあげておく．

座標変換

4行4列のマトリックスを用いれば，透視変換まで含めて表現することができる．右手系の座標系をとるか左手系の座標をとるか，原点をスクリーン上にとるか目の位置にとるか，奥行き方向をプラスにとるかマイナスにとるか，座標系を決定するだけでこれだけの選択肢があるわけで，けっこうめんどうな世界だな．

陰面処理

視点から遠いものを見えなくすること．レイトレーシングを用いれば必然的にできてしまうが，ポリゴンを用いたりするとやっかいな問題として残る．現在はメモリが安価になっているし，Zバッファ法がめんどうでなくていいと思う．すなわち，描かれた部分で視点にいちばん近いところのZ軸方向の位置情報を画面全体について保存しておき，それよりも近い点が現れたら塗り替えるし，遠い点が現れたらしらんぷりを決め込むということだ．最近のグラフィックス用ワークステーションなんかは，ハードウェアで処理しているから速いのなんのってやってられねえよ．

モデリング

3次元グラフィックスのうちで，実際に色を塗る前の変形したり移動したりという部分．ユーザーインターフェースが結構やっかいになると思う．僕が作ったトリスは，テキストファイルにだらだらとスクリプトを書く方式だった．

レンダリング

色塗りのこと．光の方向や表面の状態などを考えて（シェーディング），実際に画面に色づけしていく過程．3年ほど前のシーグラフで，ピクサー社がレンダーマンというレンダリングのた

グラフィックス用語集
カラーモード　一般にパソコンはビデオRAMを解像度優先か，色数優先に使うかで複数の表示モードを持っている．互換性を維持するために用意されていることもある．

めの標準インターフェースを公開して話題になった．最近邦訳も出たようだけれど，このマニュアルは考えるべき要素がほぼ網羅されていて，入門書としても適しているような気がする．

いずれにせよ，僕自身は3次元のレンダリングシステムを自分でインプリメントしようという気にはならない．それ専門のソフトハウスが英知を尽くしてインプリメントしているのだから，かないっこないよ．結構力仕事も多いし．どうしてもという人にまでやめろとはいわないけれど，いってみれば「一太郎」を個人で作るようなものだと思うよ．それよりも「一太郎」で何を書こうか考えたほうが建設的なような気がする．

最後に

グラフィックスのおもしろさというのは，ある原理をプログラムに置き換え，それを目でみることができる点にあるような気がする．リカーシブコールの代名詞である樹にしても，単純なプログラムで実に多彩で美しい映像を見ることができる．そして，その形自体がリカーシブコールの本質をみごとに表現している．

これからグラフィックスの世界に足を踏み入れようとしてるみなさんにも，ぜひこの醍醐味を味わっていただきたい．そのための最初で最後のコツを1つ．自分自身でやってみること．本に載っているどんなにむずかしいプログラムだろうと，どんなに美しい画像だろうと，自分自身で工夫してインプリメントした点を打つルーチンで描いた最初のプログラムほどの感動を与えてくれることはけっしてないから．

COLUMN　参考書は大きく分けて2通りある

グラフィックスの参考書は大きく分けて2通りある．1つは，ある処理系に密着して具体的なプログラムを解説したもの．たとえば，アスキーのラーニングシステムシリーズの『入門グラフィックス』（写真1）『実習グラフィックス』『応用グラフィックス』（佐藤義雄著，アスキー）や，僕と安斎利洋が書いた『ターボ・グラフィックス』（JICC出版局・写真2）など．これらは，自分の使っている環境と合致すれば，比較的容易にそのままでプログラムを動かしたり，一部改良したりして使えるので安直でよい．

しかし，一歩つっこんでプログラムをやろうとする人は，座右に置く標準的なグラフィックスの教科書をもつとよい．具体的なソースコードを読むよりも，教科書の記述を見るほうが，アルゴリズム全体に対する理解はより深まるような気がする．

僕自身の例でいくと，以下の2冊を常に参照している．DAVID F.ROGERS J.ALAN ADAMS『コンピュータグラフィックス』（山口富士夫訳，日刊工業新聞・写真3），DAVID F.ROGERS『実践コンピュータグラフィックス』（山口富士夫監修，日刊工業新聞・写真4）

前者は，座標変換や曲線や局面の数学的な記述などをしっかりと押さえてあり，10年以上前の本なのに古さを感じさせない．インプリメント例がBASICで記述されているのは残念だけれど，この手の問題は実数演算がおもになるのでインプリメントに特別なテクニックは必要としないし，さして問題とはならないだろう．

後者は前者の続編の趣があるが，より具体的にラスターグラフィックスの主要なアルゴリズムが収められており，たいへん参考になる．アルゴリズムの記述は疑似パスカル．

本文中にリストを掲げたブレゼンハムによるDDAによる円の描画やポリゴンの塗りつぶしなどは，この本を読んで自分なりにインプリメントしたものだ．

グラフィックスのプログラミングにとって，データ構造の決定は最重要な問題の1つとなる．これについても，標準的な教科書を1冊座右に置いておくとよいだろう．僕自身はパスカル教の信者でもあり，ニクラウス・ヴィルトの『アルゴリズム＋データ構造＝プログラム』（片山卓也訳，日本コンピュータ協会・写真5）をもっとも強く推す．基礎的な部分で理解しやすいのは，石畑清著『アルゴリズムとデータ構造』（岩波講座ソフトウエア科学3・写真6），阿部圭一著『ソフトウエア入門』（共立出版・写真7）などだ．

コンピュータグラフィックスで使用する数学はほとんどが高校程度，難しくて大学の教養過程程度と考えてよいだろう．大学受験用の数学公式集が意外と役に立つ．1冊手元に置いておくとよいだろう．

写真1

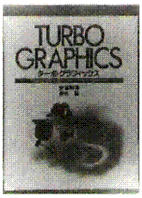
写真2


写真3

写真4

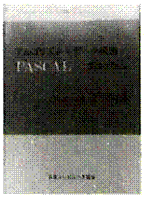
写真5


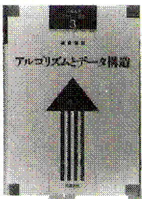
写真6


写真7

グラフィックス用語集

キャラクタジェネレータ　文字コードを文字としてディスプレイに出力するために，その文字パターンを記憶しているROMのこと．

GRAPHICS PROGRAMMING

特集

グラフィックス
プログラミング
の世界

1677万色・フルカラーグラフィックスプログラミング

フレームバッファ活用法

尾崎行雄 株式会社サピエンス

現在発売されているPC-9801シリーズで表現できる色数は，わずか16色に過ぎない．しかし，このPC-9801の拡張スロットにフレームバッファと呼ばれるグラフィックスボードを挿入することで，1677万色フルカラーの世界を堪能することができるようになる．
また，市販されているフレームバッファには，C言語またはBASICで利用可能なライブラリが付属していることが多く，これらを利用して独自のフルカラーグラフィックスプログラムを作成することが可能である．ここでは，代表的なフレームバッファであるSuperFrameの機構と，SuperFrameに付属する関数を利用したプログラミングについて解説する．

1987年は パーソナルCG元年

ここ，4～5年のうちに，パソコンで使用できるフルカラーフレームバッファが一般のユーザーにも手の届く価格で入手できるようになりました．

筆者がCGをはじめようとしたころは（はじめたわけではなかったのですが），PC-9801用では，SIGが出していたシステムがもっとも安い部類だったように記憶してます．そのころは，CGプロダクションもDECのVAX780というようなコンピュータで仕事をしていたころで，JCGLとかリンクス（当時はトーヨーリンクス）といったプロダクションのデモリールに目を輝かせていたものでした．

ホビークラスのユーザーがパソコンでなんとかフルカラーのシステムを構築できるようになったのは，1987年からです．この年は，サピエンス社のSuperFrame，デジタルアーツ社のCGフレーム98が発売されました．CGシステムでも，トーヨーリンクスのパーソナルリンクスやキャスト社のC-TRACEが発売された年です．それまで，ミニコンやスーパーコンピュータの世界でしか実現できなかったCGが，一挙にパソコンで実現できるようになりました．もちろん，スピードは何倍もかかりました．しかし，全然できないのと，遅くても可能なのには大きな差があります．現在，活躍中のCGアーティストの中には，この時期にCGをはじめた人も多いようです．ふりかえってみれば，1987年はパーソナルCG元年といえるかもしれません．

現在，パソコンで比較的よく使われているフルカラーフレームバッファは，ほとんどのものが，1画素当たりRGB各8ビット，24ビットの表示能力をもっています．各色ごとに256階調の表現ができて，全体で約1677万種類の色の表示が可能です．余談ですが，よくカタログなどに『1670万色同時表示』と銘打ったものがありますが，フレームバッファの全画素が640×400＝256,000画素しかありませんので，同時に表示することはできません．

RGB各色8ビット（256階調）というのは，かなり多いのではないかと思われる方もいるでしょう．しかしその昔，広島大学で研究用のカラーフレームバッファを業者に発注した際，最初のものは各色6ビットで，レンダリング結果を表示したところマッハバンド（グラデーションなどを表示したときに見える縞模様）が出てしまったそうです．そこで次に作るときに8ビットに拡張したところ，マッハバンドが消えたそうです．つまり，RGB各色256階調というのは自然な画像を再現するための最低線といえるのです．

SuperFrameの設計思想について

現在，もっとも一般的なアーキテクチャのフルカラーフレームバッファの元となったのはSuperFrameだといえます．SuperFrameは，対話型のフルカラーシステムを念頭において設計されています．そのため，1987年以前のフルカラーフレームバッファの常識では，考えられない設計になりました．当時の設計では，コントロールにパソコン少年憧れの的だった日立のACRTCを採用し，グラフィック端末のようにパソコンのCRTとは独立にディスプレイを必要とするタイプが主流でした．

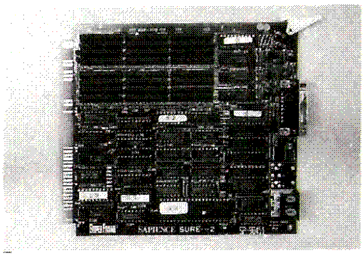


写真1：SuperFrame²Σ

そういえば，当時はまだパソコン用のアナログディスプレイがめずらしいころでした．SuperFrameは，それまでのフルカラーフレームバッファがもっていなかった3つの特徴を備えていました．それは，

- スーパーインポーズ機能
- 画素単位のアクセス
- I/Oによるアクセス

です．

では，この3つの特徴を順に説明しましょう．

スーパーインポーズ機能

フレームバッファに専用のカーソルプレーンやキャラクタプレーンなどをもつことにより，ソフトウェアのオーバーヘッドを軽減することができます．フルカラー画面に2値データあるいは8色程度の色数のカーソルを表示する

ために毎回各画素3バイトものアクセスをするのは，多くのCPUタイムを要してしまいスピードの低下を招きます．しかし，専用のプレーンを追加するのではコストが上がってしまい，製品の価格が高くなってしまいます．そこでSuperFrameでは，既存のパソコン画面とのスーパーインポーズ（重ね合わせ表示）を採用しました．カーソルやテキストは，標準で付いているパソコン側の機能を使用してしまう方法です．

これは，さらにディスプレイを1台ですませることができるという利点も生みました．それまでのフレームバッファは，パソコン用のディスプレイのほかにフレームバッファ専用のディスプレイを用意しなくてはならなかったのですが，ディスプレイが1台ですむのであれば，スペースもコストも少なくてすむわけです．

画素単位のアクセス

SuperFrameが登場するまでのフルカラーフレームバッファは，日立のACRTCといったグラフィックコントロールチップを採用していました．現在でも，アクセラレータとして高機能なチップを採用しているボードが数多く存在しています．コントローラをもつボードでは矩形領域やラインの描画を高速に処理できますが，画素単位のアクセスでは1画素描くにもたくさんのコマンドをプロセッサに送ってやる必要があるため遅くなります．

SuperFrameは，もともと対話型のCGシステムを目指して設計されたので，あえてX座標とY座標を指定して画素単位にアクセスする方式を採用したのです．

PC-9801の標準のグラフィックVRAMは，メインメモリ上にプレーン単位に配置されています．そして，このプレーン単位でアクセスする際には，各プレーンの1ビットが1画素に対応します．このため，直接VRAMを制御することで，矩形領域の転送などのブロック転送を高速に行うことが可能です．

フルカラーフレームバッファの構造は，標準のVRAMとは異なり，画素単位のアクセスが標準です．もっとも，Hyper-Frameのように，I/Oバンク方式の128KB単位のアクセスモードをもつものもありますが，画素単位の演算が主体となるレイトレーシングや画素単位にパターンを書き込むペイントシステム，また，画素をひんぱんに読み書きする画像処理ソフトなどでは，1つのコマンドで画素単位にアクセスできるほうが，画素の特定に手間のかかるプレーン方式よりソフトウェアの構築が容易になり，余分なオーバーヘッドをおさえて，高速なアクセスが可能になります．また，処理が単純になるので，バグも少なくなるでしょう．

I/O方式のアクセス

PC-9801シリーズに搭載されているi80x86CPUは，リアルモードで1MBのメモリ空間をもっています．以前は，640KBの広大なメモリ空間とい

図1

SuperFrameのレジスタ構成の概略

	D15 ………… D9 ….. D7	………………… D1
00D0	DON'T CARE	コントロールレジスタ1
00D2	DON'T CARE	コントロールレジスタ2
00D4	DON'T CARE	X軸レジスタ
00D6	DON'T CARE	Y軸レジスタ
00D8	読み出し禁止	赤レジスタ
00DA	読み出し禁止	緑レジスタ
00DC	読み出し禁止	青レジスタ
00DE	読み出し禁止	モードレジスタ

図2

X軸レジスタ，Y軸レジスタの指定範囲

X軸レジスタ　：　0～639
Y軸レジスタ　：　0～399

グラフィックス用語集
グレースケール　モノクロームなど単色で階調表示のみで表現した画像データ，またはディスプレイ．

うようなうたい文句があったくらいで，その程度のメモリですべてがたりると思われていました．

しかし，フルカラーCGを実現しようとすると，PC-9801と同様の解像度で，1画素3バイトとして，3×640×400=768,000バイトの容量が必要になります．これは，640KBのプログラム領域しかもたないMS-DOSの世界では無理な数字です．80286以降のCPUでは，1MB以上のメモリをアクセスできるようになったので，メインメモリにフレームメモリを割り当てることも可能です．しかし，フレームメモリをメインメモリに割り当てると，データの読み書きの度にXY座標から実際のアドレスを割り出すためのアドレス計算のオーバーヘッドがかかります．それゆえ，現在でもI/Oによる画素単位のアスセス方式というのは十分に通用すると思います．

フレームバッファの使い方

ここでは，仕事の都合上，筆者にもっともなじみの深いSuperFrameについてその構造を説明します．SuperFrameには，SuperFrame，SuperFrame²，SuperFrame²Σという製品があります（現在はSuperFrameは生産中止）．以下では，これらを総称してSuperFrameと呼びます．SuperFrameシリーズは，発売時期がくだるにしたがって機能が拡張されていたり，不用となった機能が外されたりしていますが，基本機能に関しては変わりがありません．ここでは，各製品の共通部分に関して説明していきます．他社のフルカラーフレームバッファも基本的なアクセス手順に関してはSuperFrameとあまり違いがないと思われますので，それぞれのマニュアルを参考にアクセスルーチンを作ることが可能だと思われます．

図3
4画素同時書き込みの例

```
書き込み画素
X=0 → 0, 1, 2 ,3
X=1 → 0, 1, 2, 3
X=2 → 0, 1, 2, 3
X=3 → 0, 1, 2, 3

X=4 → 4, 5, 6 ,7
X=5 → 4, 5, 6, 7
X=6 → 4, 5, 6, 7
X=7 → 4, 5, 6, 7
```

SuperFrameのレジスタ構成

SuperFrameは，標準の設定では0d0h～0dfhまでの16バイト分のI/Oポートを専有しています（図1）．この，I/Oポートを使用してフレームバッファへのアクセスを行います．I/Oポートのアドレスは，変更できるようになっています．しかし，ほとんどのアプリケーションがこのアドレスによってインターフェースをとっていますので，自作プログラムのため以外では，アドレスを変更するのは得策ではありません．

コントロールレジスタ

コントロールレジスタ1（0d0h）とコントロールレジスタ2（0d2h）は，ボードの初期化を行うときに使用します．初期化情報はサピエンスが供給しており，初期化の際にはこの固定値を書き込んでいきます．一般のユーザー，プログラマは，初期化時以外にはアクセスする必要のない部分です．初期化値に関してはあとで説明するアクセス関数SFInit()のリスト中に記述されています．

X軸レジスタ，Y軸レジスタ

X軸レジスタ（0d4h）とY軸レジスタ（0d6h）は，カレントポジションを指定するレジスタです．X軸レジスタは横方向の座標位置を設定するのに使用します．Y軸レジスタは縦方向の座標位置を設定します．それぞれの指定範囲は図2のようになります．

この範囲をこえた座標を指定した場合には，動作は保証されません．また，このレジスタは，ワードで書き込まないと動作しません．このポートは，書

グラフィックス用語集
誤差拡散法　ディスプレイの解像度を下げることなく多色化を達成する方法．複数の近似色を目的のドット近くに表示させ平均値による色合成を行う．

き込み専用となっていますので現在のポイントを読み出すことはできません．現在のポイントが必要な場合は，アプリケーション側で保存しておく必要があります．

色レジスタ

赤レジスタ（0d8h），緑レジスタ（0dah），青レジスタ（0dch）の3ポートが色レジスタです．これは，色情報を読み書きするためのレジスタです．光の3原色を単位とし各色256階調の指定ができます．各色の組み合わせで256(赤)×256(緑)×256(青)＝16,777,216色の色表現が可能になります．

この3つのレジスタは通常はバイトでアクセスし，値は0～255の範囲で指定できます．ポートがバイトなので，当然この値以上にはアクセスできません．しかし，アクセス方法によっては，計算値がワード以上の場合に下位8ビットのみがとられるので桁あふれがおこってしまいます．255をこえる数値は255としてアクセスするような注意が必要です．

そのほかに，4画素同時書き込み機能が用意されています．ワードでOUTした場合には，4画素バウンダリの同時書き込みが行えます．4の剰余系を先頭とする4画素に対して同時に書き込むことができます．たとえば，X座標に3を指定したときは0，1，2，3の4画素に同時に同じデータが書き込まれる，という意味です．指定したX座標から4画素ではありません．残念ながらそこまで都合よくできていないのです．おもに画面クリアのような，同一色の塗りつぶしを高速に行う場合に効果的でしょう．

モードレジスタ

モードレジスタ（0deh）は，SuperFrameの動作モードを指定するためのレジスタです．このレジスタは8ビット構成になっていて，各ビットごとに意味がもたされています．

【オートインクリメントモード】

D7とD6の組み合わせによって，X軸レジスタの自動加算ができます．X軸レジスタへの書き込みが不要となりますので，画素を順次加算していくような場合に有効です．たとえば，画像ファイルの読み込みや書き込みというような場合です．

【グレイモード】

このビットを1にすると，色レジスタに書き込んだときにほかの2色にも同様のデータが書き込まれます．

色レジスタの項目でふれた，4画素同時書き込み機能と併用することにより，高速にグレイを書き込むことができます．書き込む値を0に設定すれば，高速画面消去が実現できます．

【オーバーレイモード】

フルカラー画面を表示するかどうかを決定するビットです．1にするとフルカラー画面を表示します．なお，パソコン画面の非表示機能については，BIOS（18h）の使用を推奨しています．

サンプルプログラム

今回SuperFrameシリーズ用に作成したプログラムは，すべてをC言語で記述しました．これまで製品に添付されていた関数群では，アセンブリ言語によるC言語の関数として記述されていました．しかし，これではメモリモデルによって，複数のライブラリを用意しなければならず，リンク時にも注意が必要ですし，メンテナンスも結構めんどうです．そこで，Cで記述してもある程度のパフォーマンスを得るために，I/O命令の部分をインラインで展開して，関数呼び出しを極力避けるような構成としました．

アセンブリ関数を利用すると，一見速度が速くなるように思われますが，

図4
モードレジスタ

D7，D6	：	オートインクリメントモード
D3	：	グレイモード
D0	：	オーバーレイモード

図5
オートインクリメントモード

7	6		
0	0	：	オートインクリメント解除モード
0	1	：	リード，ライト時にオートインクリメント
1	0	：	リザーブ（使用不可）
1	1	：	ライト時のみオートインクリメント

グラフィックス用語集
コンピュータグラフィックス　コンピュータによる図形処理全般をいう．処理速度が大幅に改善されパソコンでも手軽にグラフィックスを扱うことができるようになった．

呼び出しがひんぱんになる画素単位の読み書きでは，毎回関数をコールすることになりますので，かえってオーバーヘッドがかかります．

このサンプルは，Microsoft C Compiler Version 6.00A (以下MS-C)で記述されていますが，5.1でも十分使用できます．MS-Cでは，コンパイル時に/Oiオプションを指定することにより，I/O関数のinp()，inpw()，outp()，outpw()をインライン展開にします．つまり，関数を呼ぶのではなく，直接IN，OUT命令を発行するコードを出力します．これにより，関数呼び出しのオーバーヘッドがさけられ，1画素ごとの出力に関してアセンブリルーチンを呼び出すより高速に処理をすることができます．また，これらの低水準のアクセスのために関数型マクロを使用しているため，純粋なCプログラマにとっては可読性が高く，変更も容易になっています（と，思います）．

このほかに，MS-C V.6.0でようやく導入されたインラインアセンブル記述による方法もありますが，関数型マクロの中にインラインアセンブラで記述した場合に，予期しないオブジェクトを生成したり，C言語としての可読性が低くなることも考えられます．また，展開コードをみても，インラインで展開したものとあまり差があるとは思えません．

マクロで定義した機能

これから紹介するのは，マクロとして記述した機能です(表1)．これらの機能は，非常にプリミティブで比較的

リスト1　1画素書き込みマクロ

```
#define SFPset(x,y,r,g,b)        {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){ ¥
                                        outpw(XREG,x);          ¥
                                        outpw(YREG,y);          ¥
                                        outp(REDREG,r);         ¥
                                        outp(GRNREG,g);         ¥
                                        outp(BLUREG,b);}}
```

リスト2　1画素書き込みマクロ /Oi 展開オブジェクト

```
        cmp     WORD PTR _x,0
        jl      $1172
        cmp     WORD PTR _x,640
        jge     $1172
        cmp     WORD PTR _y,0
        jl      $1172
        cmp     WORD PTR _y,400
        jge     $1172
        mov     ax,WORD PTR _x   ; outpw(212, x);
        out     212, ax

        mov     ax,WORD PTR _y   ; outpw(212, y)
        out     214, ax

        mov     ax,WORD PTR _r   ; outp(216, r)
        out     216, al

        mov     ax,WORD PTR _g   ; outp(218, g)
        out     218, al

        mov     ax,WORD PTR _b   ; outp(220, b)
        out     220, al

$1172:
```

リスト3　1画素書き込みマクロ関数呼びだしオブジェクト

```
        cmp     WORD PTR _x,0
        jl      $1115
        cmp     WORD PTR _x,640
        jge     $1115
        cmp     WORD PTR _y,0
        jl      $1115
        cmp     WORD PTR _y,400
        jge     $1115
        push    WORD PTR _x      ; outpw(212,x)
        mov     ax,212
        push    ax
        call    _outpw
        add     sp,4
        push    WORD PTR _y      ; outpw(214,y)
        mov     ax,214           ;
        push    ax               ;
        call    _outpw           ;
        add     sp,4
        push    WORD PTR _r      ; outp(216,r);
        mov     ax,216
        push    ax
        call    _outp
        add     sp,4
        push    WORD PTR _g      ; outp(218,g);
        mov     ax,218
        push    ax
        call    _outp
        add     sp,4
        push    WORD PTR _b      ; outp(220,b);
        mov     ax,220
                push    ax
                call    _outp
                add     sp,4
$1115:
```

グラフィックス用語集
座標　空間内での絶対位置を示すための値．ディスプレイ上では一般的に左上端を原点として右方向にX軸の正方向，下方向にY軸の正方向が定義されている．

リスト4 SFmacro.h

```
/*==========================================================================*/
/*                            S A P I E N C E                               */
/*--------------------------------------------------------------------------*/
/*                            SFmacro.h                                     */
/*--------------------------------------------------------------------------*/
/*                    ◎SuperFrameにアクセスするためのマクロ                 */
/*                    ◎アクセス関数のプロトタイプ                           */
/*                    ◎アクセスのための定数定義                             */
/*--------------------------------------------------------------------------*/
/*  Copyright (C) SAPIENCE Corp. 1991. All rights reserved.                 */
/*                                 written by Peco.                         */
/*==========================================================================*/

/* ++++++++ 定数定義 ++++++++ */

/* SuperFrame I/Oアドレス          */
#define CONT0       0xd0
#define CONT1       0xd2
#define XREG        0xd4
#define YREG        0xd6
#define REDREG      0xd8
#define GRNREG      0xda
#define BLUREG      0xdc
#define MODREG      0xde

/* SuperFrame 画面サイズ            */
#define SFXSIZE     640
#define SFYSIZE     400

/* BOOL definition  */
#define FALSE   0
#define TRUE    1

/* ++++++++ SuperFrame アクセス用マクロ ++++++++ */

/*======================================================================*/
/*              SFRedSet(x, y, r)                                       */
/*----------------------------------------------------------------------*/
/*          指定の座標に赤を設定します                                  */
/*              x : X座標                                               */
/*              y : Y座標                                               */
/*              r : 色(0-255)                                           */
/*======================================================================*/
#define SFRedSet(x,y,r)  {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                           outpw(XREG,x);       ¥
                           outpw(YREG,y);       ¥
                           outp(REDREG, r);}


/*======================================================================*/
/*              SFGreenSet(x, y, g)                                     */
/*----------------------------------------------------------------------*/
/*          指定の座標に緑を設定します                                  */
/*              x : X座標                                               */
/*              y : Y座標                                               */
/*              g : 色(0-255)                                           */
/*======================================================================*/
#define SFGreenSet(x,y,g)    {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                           outpw(XREG,x);       ¥
                           outpw(YREG,y);       ¥
                           outp(GRNREG,g);}

/*==========================================================================*/
/*              SFBlueSet(x, y, b)                                          */
/*--------------------------------------------------------------------------*/
/*          指定の座標に青を設定します                                      */
/*              x : X座標                                                   */
/*              y : Y座標                                                   */
/*              b : 色(0-255)                                               */
/*==========================================================================*/
#define SFBlueSet(x,y,b)     {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                           outpw(XREG,x);       ¥
                           outpw(YREG,y);       ¥
                           outp(BLUREG,b);}

/*==========================================================================*/
/*              SFPset(x, y, r, g, b)                                       */
/*--------------------------------------------------------------------------*/
/*          指定の座標に指定色を設定します                                  */
/*              x : X座標                                                   */
/*              y : Y座標                                                   */
/*              r : 色(0-255)                                               */
/*              g : 色(0-255)                                               */
/*              b : 色(0-255)                                               */
/*==========================================================================*/
#define SFPset(x,y,r,g,b)    {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){ ¥
                           outpw(XREG,x);       ¥
                           outpw(YREG,y);       ¥
                           outp(REDREG,r);      ¥
                           outp(GRNREG,g);      ¥
                           outp(BLUREG,b);}}

/*==========================================================================*/
/*              SFPsetGray(x, y, c)                                         */
/*--------------------------------------------------------------------------*/
/*          指定画素に指定色（グレー）を設定します                          */
/*              x : X座標                                                   */
/*              y : Y座標                                                   */
/*              c : 色(0-255)                                               */
/*==========================================================================*/
#define SFPsetGray(x,y,c)  {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){ ¥
                           outpw(XREG,x);       ¥
                           outpw(YREG,y);       ¥
                           outp(REDREG,c);      ¥
                           outp(GRNREG,c);      ¥
                           outp(BLUREG,c);}}

/*==========================================================================*/
/*              SFMaxPset(x, y, r, g, b)                                    */
/*--------------------------------------------------------------------------*/
/*          指定画素にの座標に指定色を設定します。                          */
/*          現在の色より(r,g,b)の値が大きいときのみ設定します。             */
/*              x : X座標                                                   */
/*              y : Y座標                                                   */
/*              r : 色(0-255)                                               */
/*              g : 色(0-255)                                               */
/*              b : 色(0-255)                                               */
/*==========================================================================*/
#define SFMaxPset(x,y,r,g,b)  {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){ ¥
                           outpw(XREG,x);           ¥
                           outpw(YREG,y);           ¥
                           if(inp(REDREG)<=r){      ¥
```

使用頻度が高いと思われます．SFmacro.hというヘッダに記述されていますので，プログラムの先頭でこのヘッダのインクルードを記述しておけば使用できます（リスト4）．ただ，マクロなので副作用をもつ変数を引数として渡す場合などには，思い通りに動作しない可能性があります．マクロであるということを肝に命じて使用しなくてはなりません．

機能としては，それぞれ次の3つの処理を行うようになっています．

クリッピング処理

フレームバッファの範囲外に書き込まないよう，クリッピング処理をかけています．SuperFrameでは，フレームバッファの範囲外の画素を指定した場合は，動作を保証していません．必ず範囲内しか書き込まないという保証があれば，先頭のif文は外してしまってかまわないでしょう．その分スピードが向上します．

座標の設定

X座標とY座標を設定します．

画素の色情報のやり取り

ここで処理がわかれます．それぞれの関数の機能に応じて色情報のI/Oを行います．

なお，この関数マクロでは変数型のキャストをかけていないので，指定以外の変数では動作が保証されません．

C言語関数として実現した機能

表2の関数は，Cで書き起こしたものです．SuperFrameとのI/Oの部分は前述のマクロを使用しています．このようにインライン展開のオプションをつけた場合は，下位のアクセス関数をアセンブリで記述された場合より高速に動作することもあります（リスト5，6）．

ライン描画ルーチン(SFLINE.C)

ラインの描画を行う関数に，このマクロを組み込んでみます（リスト7）．

グラフィックス用語集
サーフェイスモデル　3次元グラフィックスなどで物体を面（サーフェイス）の集合として表現する方法．陰線処理や陰面処理によって物体の立体感が得られる．

```
                                outp(REDREG,r);}      ¥
                            if(inp(GRNREG)<=g){       ¥
                                outp(GRNREG,g);}      ¥
                            if(inp(BLUREG)<=b){       ¥
                                outp(BLUREG,b);}}}

/*=====================================================================*/
/*          SFMinPset(x, y, r, g, b)                                   */
/*---------------------------------------------------------------------*/
/*      指定画素にの座標に指定色を設定します。                           */
/*      現在の色より(r,g,b)の値が小さいときのみ設定します。               */
/*          x : X座標                                                  */
/*          y : Y座標                                                  */
/*          r : 色(0-255)                                              */
/*          g : 色(0-255)                                              */
/*          b : 色(0-255)                                              */
/*=====================================================================*/
#define SFMinPset(x,y,r,g,b) {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){ ¥
                            outpw(XREG,x);            ¥
                            outpw(YREG,y);            ¥
                            if(inp(REDREG)>r){        ¥
                                outp(REDREG,r);}      ¥
                            if(inp(GRNREG)>g){        ¥
                                outp(GRNREG,g);}      ¥
                            if(inp(BLUREG)>b){        ¥
                                outp(BLUREG,b);}}}

/*=====================================================================*/
/*          SFRedGet(x, y, r)                                          */
/*---------------------------------------------------------------------*/
/*      指定画素の赤の値を取得します                                     */
/*          x : X座標                                                  */
/*          y : Y座標                                                  */
/*          r : 色(0-255)                                              */
/*=====================================================================*/
#define SFRedGet(x,y,r) {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                            outpw(XREG,x);        ¥
                            outpw(YREG,y);        ¥
                            r=inp(REDREG);}}

/*=====================================================================*/
/*          SFGreenGet(x, y, g)                                        */
/*---------------------------------------------------------------------*/
/*      指定画素の緑の値を取得します                                     */
/*          x : X座標                                                  */
/*          y : Y座標                                                  */
/*          g : 色(0-255)                                              */
/*=====================================================================*/
#define SFGreenGet(x,y,g)   {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                            outpw(XREG,x);        ¥
                            outpw(YREG,y);        ¥
                            g=inp(GRNREG);}}

/*=====================================================================*/
/*          SFBlueGet(x, y, b)                                         */
/*---------------------------------------------------------------------*/
/*      指定画素の青の値を取得します                                     */
/*          x : X座標                                                  */
/*          y : Y座標                                                  */
/*          b : 色(0-255)                                              */
/*=====================================================================*/
#define SFBlueGet(x,y,b)    {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                            outpw(XREG,x);        ¥
                            outpw(YREG,y);        ¥
                            b=inp(BLUREG);}}

/*=====================================================================*/
/*          SFGet(x, y, r, g, b)                                       */
/*---------------------------------------------------------------------*/
/*      指定画素の青の値を取得します                                     */
/*          x : X座標                                                  */
/*          y : Y座標                                                  */
/*          r : 色(0-255)                                              */
/*          g : 色(0-255)                                              */
/*          b : 色(0-255)                                              */
/*=====================================================================*/
#define SFGet(x,y,r,g,b)    {if(((x>=0)&&(x<640))&&((y>=0)&&(y<400))){  ¥
                            outpw(XREG,x);        ¥
                            outpw(YREG,y);        ¥
                            r=inp(REDREG);        ¥
                            g=inp(GRNREG);        ¥
                            b=inp(BLUREG);}}


/*----- 関数のプロトタイプ -----*/
void SFInit( int, int );
void SFCircle(int, int, unsigned int,
              unsigned char, unsigned char, unsigned char);
void SFLine(int, int, int, int, unsigned char, unsigned char, unsigned char);
void SFBoth( void );
void SFPConly( void );
void SFonly( void );
void SFclr( void );

/*----- InitSF() 関数の第1引数の定数 -----*/
#define BOTH        0x01        /* PC画面とSF画面の両方を表示する    */
#define PCONLY      0x00        /* PC画面のみを表示する              */
#define SFONLY      0x03        /* SF画面のみを表示する              */
#define NOCHANGE    0xff        /* 現状を維持する                    */

/*----- InitSF() 関数の第2引数の定数 -----*/
#define CLR         1       /* スーパーフレームをクリアする          */
#define RSV         0       /* スーパーフレームをそのままの状態にする  */


typedef     struct
                {
                unsigned char   r;
                unsigned char   g;
                unsigned char   b;
                }   COLOR;

typedef     struct
                {
                int     x;
                int     y;
                }   POINT;


/*===== end of sfmacro.h =====*/
```

このライン描画関数は，SuperFrameのパッケージに標準添付されているライブラリのものを元にしています．フレームバッファへの描画の部分のみを関数マクロを使うように変更しました．

プログラム自体は，標準的なDDAアルゴリズムで書かれています．本稿は，フレームバッファの使い方ということで，アルゴリズムに関してはほかのくわしい参考書を参考にしてください．参考文献4）に掲載されているグラフィックス関数を参考にして表示部分を書き換えるだけで，円，楕円などの描画が可能になります．

閉領域の塗りつぶし（SFFILL.C）

閉領域を塗りつぶす関数を作ってみます．基本的なロジックは参考文献6）のBASICで書かれたものを参考にしています．機能はBASICに用意されていたPAINT文と同様で，指定された境界色の中を塗りつぶすものです．

ここでは，SuperFrameにアクセスする方法として，2種類の方法を用いています．1つめはマクロを使用する方法です．もう1つは，outp()関数を使用して直接アクセスする方法です．マクロは便利ですが，機能的には限られたことしか実現できません．細かい処理を高速に行おうとすると，やはり，直接アクセスする関数を定義するほうが効率がいいでしょう．

指定座標の色を境界色と比較するための関数TestSF()を定義しています．これは，与えた色値とフレームバッファ上の色値を比較してその結果を返す関数です．3色を次々に比較して，一致しなかった場合にはFALSEで終了します．すべてが一致した場合はTRUEを返します．マクロで定義されているSFPget()を使用すると，一度に3色をとってしまいます．SFPgetRed(), SFPgetGreen(), SFPgetBlue()を順次使用すると，呼び出すごとにX，Y座標のセットを行ってしまいます．どちらの選択でも“帯に短し，たすきに長し”といったところでしょう．

このサンプルは単純な機能のみをサポートしているので，

①境界色に一定の幅をもたせる
②塗りつぶし色をパターンやグラデ

表1　マクロで定義した機能

関数	説明
SFRedSet(x, y, r)	指定の座標に赤データを設定します. x：X座標値 y：Y座標値 r：赤色の値
SFGreenSet(x, y, g)	指定の座標の緑プレーンに色を設定します. x：X座標値 y：Y座標値 g：緑色の値
SFBlueSet(x, y, b)	指定の座標の青プレーンに色を設定します. x：X座標値 y：Y座標値 b：青色の値
SFPset(x, y, r, g, b)	指定の座標に赤, 緑, 青データを設定します. x：X座標値 y：Y座標値 r：赤色の値 g：緑色の値 b：青色の値
SFMaxPset(x, y, r, g, b)	指定座標の現在の色が指定色より小さい場合に色をセットします. x：X座標値 y：Y座標値 r：赤色の値 g：緑色の値 b：青色の値 指定座標の色データを取得しながら与えられた色データと比較する処理が付け加えられています.
SFMinPset(x, y, r, g, b)	指定座標の現在の色が指定色より大きい場合に色を設定します. x：X座標値 y：Y座標値 r：赤色の値 g：緑色の値 b：青色の値
SFRedGet(x, y, r)	指定の座標の赤の色情報を取得します. x：X座標値 y：Y座標値 r：赤色の値が格納されて返ってくる
SFGreenGet(x, y, g)	指定の座標の緑の色情報を取得します. x：X座標値 y：Y座標値 g：緑色の値が格納されて返ってくる
SFBlueGet(x, y, b)	指定の座標の青の色情報を取得します. x：X座標値 y：Y座標値 b：青色の値が格納されて返ってくる
SFGet(x, y, r, g, b)	指定の座標の赤, 緑, 青の色情報を取得します. x：X座標値 y：Y座標値 r：赤色の値が格納されて返ってくる g：緑色の値が格納されて返ってくる b：青色の値が格納されて返ってくる

ーションにする

などの拡張が考えられます.

パソコン標準のグラフィックスでは, 8種類あるいは16種類の色判定で塗りつぶしができますが, フルカラー情報は色の幅が広いため, 目で見たときに色が同じようでもデータ上は違っている可能性があります. そこで, ①のように色の判定で境界色の範囲を広げて塗りつぶしを確実に行うということが考えられます.

国内で入手できるフレームバッファ製品紹介

これまで述べてきたようなI/O方式のフレームバッファでは, Super Frame以外ではデジタルアーツ社のHyper-Frame+とCast社の販売するフルカラーフレームバッファ「写像」がポピュラーで比較的入手しやすいと思われます.

とくにHyper-Frame+は, 対応したフリーウェアが数多くBBSなどに流されているようです. また, マウスプレーン, ウィンドウ機能やハードウェアズーム, スムーズスクロール機能といった工夫が凝らされています. 残念ながら筆者は, この商品に触れたことがないのでカタログスペック以上のコメントはできません.

写像は, C-TRACEのアニメ機能を使えばアニメーションのプレビューが可能です. また, 普通のフルカラーフレームバッファのサイズは640×400ですが, 写像ではフレームサイズが

グラフィックス用語集
GDC(Graphic Display Controller)　PC-9800シリーズに内蔵されるグラフィックス描画用LSI. このLSIに直接コマンドを送ることにより高速に直線や円弧を描くことができる.

640×512と縦方向が大きくなっています(表示は640×400)．これは，設計当時にVTR（NTSC）への出力を想定していたためですが，残念ながらVTR出力のオプションは現在はサポートされていません．また，写像もハードウェアズームやスクロール機能を備えています．

プログラマ向けの雑誌でこんなことを書くのも変ですが，フルカラーフレームバッファの機種選定には，ソフトウェアが大きな比重をしめているようです．あのソフトが使いたいからという理由で，どのフレームバッファを選ぶかが決るようです．逆にいえば，フレームバッファの機能と描画速度などの性能をよく評価したうえで，これから書くソフトウェアに，どのフレームバッファが適しているかを検討して選定することが肝要です．

たとえば，いままで述べてきたようなタイプのフレームバッファは画素単位の操作をする仕事には向いていますが，矩形領域をはげしく移動するような用途には向いているとはいえません．適当でない素材を料理するのは，苦労ばかり多くて実りが少ないものです．

対話型のシステムを構築するならば，やはりキーとなるのは処理速度だと思います．ペイントシステムでは，ユー

リスト5　SFInit.c

```
#include    <dos.h>
#include    "sfmacro.h"
/*=======================================================================*/
/*                          S A P I E N C E                              */
/*-----------------------------------------------------------------------*/
/*        SuperFrame 2  : access functions                               */
/*                                                                       */
/*  SuperFrameの初期化に関する関数群                                     */
/*  Microsoft C V6.00Aによって記述されています。                         */
/*  コンパイル時には、インライン展開最適化オプション( /Oi )を指定して   */
/*  ください。                                                           */
/*-----------------------------------------------------------------------*/
/*  Copyright (C) SAPIENCE Corp. 1991. All rights reserved.              */
/*                              written by Peco.                         */
/*=======================================================================*/

/*--------- SuperFrame 初期化情報 ---------*/
#define INITDATASIZE 15
static unsigned char InitData[]=
    {
    0x00, 0x6b, 0x01, 0x50, 0x02, 0x5b, 0x03, 0X0e,
    0x04, 0x1a, 0x05, 0x02, 0x06, 0x19, 0x07, 0x19,
    0x08, 0x80, 0x09, 0x0f, 0x0c, 0xff, 0x0d, 0xff,
    0x1b, 0x01, 0x1e, 0x8c, 0x1f, 0x00
    };

/*---------- 割り込み用レジスタ構造体 ----------*/
static  union REGS InRegs, OutRegs;
static  int mod;

/*=======================================================================*/
/*                SFInit( int mode, int clearsf);                        */
/*-----------------------------------------------------------------------*/
/*        SuperFrame 2/2Σの初期化処理                                   */
/*-----------------------------------------------------------------------*/
/*              mode    : 表示モード                                     */
/*              clearsf : 画面の初期化                                   */
/*=======================================================================*/
void
SFInit(int  mode, int clearsf)
    {
    int i,j;
    unsigned char *InitDatap=InitData;

    for(i=0; i<INITDATASIZE; i++)       /* SuperFrameの初期化情報を転送する */
        {
        outp(CONT0,*(InitDatap++));
        outp(CONT1,*(InitDatap++));
        }

    if( clearsf == CLR )                /* 画面の初期化をする       */
        {                               /* 4画素同時書き込みモード  */
        for(i=0;i<SFXSIZE;i+=4)
            {
            outpw(XREG,i);
            for(j=0;j<SFYSIZE;j++)
                {
                outpw(YREG,j);
                outpw(REDREG,0x00);
                }
            }
        }

    switch( mode )              /* 表示モードによる処理     */
        {
        case NOCHANGE:          /* 変化なし                 */
            outp(MODREG, 0);
            break;
        case PCONLY:            /* PC画面のみ表示する       */
            outp(MODREG, 0);
            InRegs.h.ah=0x0c;
            int86(0x18, &InRegs, &OutRegs);
            break;
        case BOTH:              /* SF画面とPC画面の両方を表示する   */
            outp(MODREG, 0x01);
            break;
        case SFONLY:            /* SF画面のみを表示する             */
            InRegs.h.ah=0x0d;
            int86(0x18, &InRegs, &OutRegs);
            InRegs.h.ah=0x41;
            int86(0x18, &OutRegs, &OutRegs);
            break;
        }
    }

/*=======================================================================*/
/*        void SFBoth( void )                                            */
/*-----------------------------------------------------------------------*/
/*              SuperFrmaeとPC画面の両方を表示する                       */
/*=======================================================================*/
void
SFBoth()
    {
    outp(MODREG, 0x01);
    InRegs.h.ah=0x0c;
    int86(0x18, &InRegs, &OutRegs);
    }

/*=======================================================================*/
/*        void SFPConly( void )                                          */
/*-----------------------------------------------------------------------*/
/*              PC画面のみを表示する                                     */
/*=======================================================================*/
void
SFPConly()
    {
    outp(MODREG, 0);
    InRegs.h.ah=0x0c;
    int86(0x18, &InRegs, &OutRegs);
    }

/*=======================================================================*/
/*        void SFonly( void )                                            */
/*-----------------------------------------------------------------------*/
/*              SF画面のみを表示する                                     */
/*=======================================================================*/
void
SFonly()
    {
    unsigned char   Mode;

    InRegs.h.ah=0x0d;
    int86(0x18, &InRegs, &OutRegs);
    InRegs.h.ah=0x41;
    int86(0x18, &OutRegs, &OutRegs);

    Mode = 0x01;
    outp(MODREG, Mode);
    }

/*=======================================================================*/
/*        void SFClr( void )                                             */
/*-----------------------------------------------------------------------*/
/*              SF画面を黒(0,0,0)で消去する                              */
/*=======================================================================*/
void
SFClr()
    {
    int     i,j;
    for(i=0;i<640;i+=4)
        {
        outpw(XREG,i);
        for(j=0;j<400;j++)
            {
            outpw(YREG,j);
            outpw(REDREG,0x00);
            }
        }
    }

/* ----- end of initsf.c ----- */
```

グラフィックス用語集

シェーディング　レイトレーシングで物体をディスプレイに表示するとき，光源に対する影を付ける処理．これによって物体の立体感を高めることができる．

リスト 6　SFCircle.c

```
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*:      ++++SFCircle.c++++                                       :*/
/*:                                                               :*/
/*:          circle function(high speed)                          :*/
/*:          (C)copyright SAPIENCE CORP. 1987,1991                :*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
#include    "sfmacro.h"

void
SFCircle(x0,y0,rr,r,g,b)
    int                 x0 , y0 ;
    unsigned  int       rr ;
    unsigned  char      r , g , b ;

    { int    x , y ;
      int    d ;

      y = rr ;
      d = 3 - 2*rr ;

      for (x=0 ; x<=y ; x++)
            { SFPset(x0+x,y0+y,r,g,b) ;
              SFPset(x0-x,y0+y,r,g,b) ;
              SFPset(x0+x,y0-y,r,g,b) ;
              SFPset(x0-x,y0-y,r,g,b) ;
              SFPset(x0+y,y0+x,r,g,b) ;
              SFPset(x0-y,y0+x,r,g,b) ;
              SFPset(x0+y,y0-x,r,g,b) ;
              SFPset(x0-y,y0-x,r,g,b) ;

              if  (d<0)       d += (x<<2) + 6 ;
                  else      { d += ((x-y)<<2) + 10 ;
                              y -- ; }
            }

    }
```

ザーは紙に書くようなスピードをコンピュータに要求します．高速な処理速度を実現するには，よく考えられた名人芸的なテクニックが必要になるかもしれません．そういったことを追求していくのも，苦しくも楽しいプログラミングですね．

いろいろ書いてきましたが，不備な点も多いかと思います．つたない文章とプログラムではありますが，みなさんがフルカラーフレームバッファをドライブして素敵なソフトを書くうえで，少しでも参考になれば幸いです．

※本章に掲載されたプログラムの著作権は，株式会社サピエンスに所属しています．

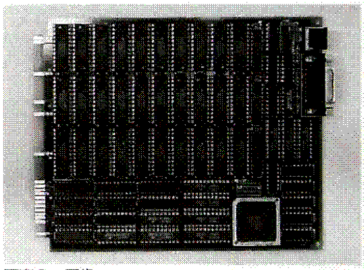


写真 2 ：写像

表 2　C言語関数として実現した機能

void SFInit(int mode, int clearsf);
SuperFrameの初期化を行います．

mode：表示モードの選択　以下のシンボルはSFMACRO.Hで定義されています．
- NOCHANGE：変化しない
- PCONLY：パソコン画面のみを表示する．
- SFONLY：SuperFrame画面のみを表示する．
- BOTH：パソコン画面とSuperFrame画面をインポーズして表示

clearsf：初期化時にSuperFrameをクリアするかどうかを指定します．次のシンボルはSFMACRO.Hに記述されています．
- CLR：クリアする
- RSV：クリアしない

void SFCircle(int x0, int y0, int rr, unsigned char r, unsigned char g, unsigned char b);
DDAアルゴリズムによって円を描きます．

- x0：中心のX座標
- y0：中心のY座標
- rr：円の半径
- r：赤の値
- g：緑の値
- b：青の値

void SFLine(int x0,int y0,int x1,int y1, unsigned char r,unsigned char g,unsigned char b);
ラインを引きます．

- x0：線の開始X座標
- y0：線の開始Y座標
- x1：線の終了X座標
- y1：線の終了Y座標
- r：赤の値
- g：緑の値
- b：青の値

void SFBoth(void);
PC画面とSF画面をインポーズ表示します．

void SFPConly(void);
PC画面のみを表示します．

void SFonly(void);
SF画面のみを表示します．

void SFclr(void);
SF画面を0で塗りつぶします．

グラフィックス用語集
ジャギー　スクリーンに表れる画素のギザギザ．

リスト7 SFLine.c

```
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*:        ++++line.c++++                                     :*/
/*:                                                           :*/
/*:           line function                                   :*/
/*:           (C)copyright SAPIENCE CORP. 1987,1991           :*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
#include    "sfmacro.h"

void
SFLine(x0,y0,x1,y1,r,g,b)
    int      x0 , y0 , x1 , y1 ;
    unsigned char    r , g , b ;
    { int    dx , adx ;
      int    dy , ady ;
      int    i , j ;
      int    i0 , i1 , di , j0 , dj ;
      int    d , d0 , d1 , adj ;

      dx = x1 - x0 ;  dy = y1 - y0 ;

      if  (dx>0)  adx =  dx ;
          else    adx = -dx ;
      if  (dy>0)  ady =  dy ;
          else    ady = -dy ;

      if  (adx>ady)
          { if  (dx>0)  { i0 = x0 ; i1 = x1 ; j = y0 ; dj =  dy ; }
                else    { i0 = x1 ; i1 = x0 ; j = y1 ; dj = -dy ; }

            d0 = ady + ady ;  d1 = d0 - adx - adx ;

            d = -adx ;

            if  (dj>0)
                for (i=i0 ; i<=i1 ; i++)
                    { if  (d<0)   d += d0 ;
                          else  { d += d1 ;  j++ ; }
                      SFPset(i,j,r,g,b) ;
                    }
            else
                for (i=i0 ; i<=i1 ; i++)
                    { if  (d<0)   d += d0 ;
                          else  { d += d1 ;  j-- ; }
                      SFPset(i,j,r,g,b) ;
                    }
          }
      else
          { if  (dy>0)  { i0 = y0 ; i1 = y1 ; i = x0 ; di =  dx ; }
                else    { i0 = y1 ; i1 = y0 ; i = x1 ; di = -dx ; }

            d0 = adx + adx ;  d1 = d0 - ady - ady ;

            d = -ady ;

            if  (di>0)
                for (j=i0 ; j<=i1 ; j++)
                    { if  (d<0)   d += d0 ;
                          else  { d += d1 ;  i++ ; }
                      SFPset(i,j,r,g,b) ;
                    }
            else
                for (j=i0 ; j<=i1 ; j++)
                    { if  (d<0)   d += d0 ;
                          else  { d += d1 ;  i-- ; }
                      SFPset(i,j,r,g,b) ;
                    }
          }

    }
```

リスト8 SFFill.c

```
#include    "sfmacro.h"

/* static functions Prototypes  */
static  int     TestColor(COLOR, COLOR);
static  int     SFTest(int, int, COLOR);

static  POINT   Stack[1000];

void
SFFill(POINT SeedPoint, COLOR b, COLOR c)
    {
    int      Point=0;
    int      Up, UpLast, Down, DownLast;
    int      x, y;
    COLOR    PointUp, PointDown;

#ifdef DEBUG
printf("x,y=%d,%d ",SeedPoint.x,SeedPoint.y);
printf("b:%d,%d,%d  ",b.r,b.g,b.b);
printf("c:%d,%d,%d¥n",c.r,c.g,c.b);
#endif

    Stack[Point] = SeedPoint;
    while( Point > -1 )
        {
        x = Stack[Point].x;     y = Stack[Point].y;
        Point--;
        if( SFTest(x, y, b) || SFTest(x, y, c) )    continue;
        while( x>=0 && !SFTest(x, y, b) )   x--;
        x++;
        Up = Down = 0;                  /* Clear Up and Down Flag   */
        while((x < SFXSIZE-1) && (!SFTest(x,y,b)) )
            {
            SFPset(x, y, c.r, c.g, c.b );
            UpLast = Up;                /* Save Last Up Flag    */
            if ( y > 0 )
                {
                SFGet(x, y-1, PointUp.r, PointUp.g, PointUp.b);
                }
            Up = ( TestColor(PointUp, b) || TestColor(PointUp, c) ) ? 0 : 1;
            if( (UpLast == 0) && (Up == 1) )
                {
                Point++;
                Stack[Point].x = x;
                Stack[Point].y = y-1;
                }

            DownLast=Down;              /* Save Last Down Flag  */
            if( y < SFYSIZE-1 )
                {
                SFGet(x,y+1,PointDown.r, PointDown.g, PointDown.b);
                }
            Down = (TestColor(PointDown,b)||TestColor(PointDown,c)) ? 0 : 1;
            if( (DownLast == 0) && (Down == 1) )
                {
                Point++;
                Stack[Point].x = x;
                Stack[Point].y = y+1;
                }
            x++;
            }
        ;
        }
    }

/*--------------------------------------------------------------*/
/*      ピクセルと色情報の比較                                    */
/*--------------------------------------------------------------*/
static  int
SFTest( int x, int y, COLOR c )
    {
                    /* フレームの範囲を越えた場合はマッチングしたものとする */
    if(( x < 0 ) || (x >= SFXSIZE ))
        return TRUE;
    if(( y < 0 ) || (y >= SFYSIZE ))
        return TRUE;
    outpw(XREG, x);
    outpw(YREG, y);
    if( inp(REDREG) != c.r )    return FALSE;
    if( inp(GRNREG) != c.g )    return FALSE;
    if( inp(BLUREG) != c.b )    return FALSE;
    return TRUE;
    }

/*--------------------------------------------------------------*/
/*      色情報の比較                                              */
/*--------------------------------------------------------------*/
static  int
TestColor(COLOR s, COLOR d)
    {
    if(s.r != d.r )     return FALSE;
    if(s.g != d.g )     return FALSE;
    if(s.b != d.b )     return FALSE;
    return TRUE;
    }
```

参考文献

1）サピエンス：SuperFrame²/²Σ ソフトウェア編，サピエンス，1990

2）サピエンス：SueprFrame²Σ 基礎・ハードウェア編，サピエンス，1990

3）蓮池曜：最新のフレームメモリの構造と使い方，PIXEL，No.61，p.87-p.88

4）奥村晴彦：C言語による最新アルゴリズム事典，技術評論社，1991

5）長島忍：CGのための図学（6），PIXEL，No.79，p.154-p.159

6）千葉，村岡，小沢，海野：コンピュータアルゴリズム全科，啓学出版，1991

グラフィックス用語集

シミュレーション　実物を使って実現できない実験を，模擬的に実験すること．処理速度の向上により，よりリアルなグラフィックスを使った実験ができるようになった．

GRAPHICS PROGRAMMING

特集

グラフィックス
プログラミング
の世界

超リアルな画像作成テクニック

レイトレーシング入門

稲葉正彦

メディアスタジオ（株）

3次元コンピュータグラフィックスを作成する際には，まずモデリングソフトウェアで物の形や配置を決定し，さらにレンダリングソフトウェアを用いて画像計算を行う．レンダリングには数多くの手法が存在しているが，ここでは，その中からとくにパーソナルコンピュータ上で実現しやすく，かつ超リアルな画像制作が可能なレイトレーシング法をとりあげる．直接プログラムは示さないが，本誌の読者であればここにあげた数式からプログラムを作成するのは容易なことだろう．

はじめに

3次元コンピュータグラフィックスは，以前にくらべるとかなり身近な映像ツールになってきた．安価なハードウェアが登場し，充実した機能をもつコンピュータグラフィックス・ソフトウェアが市販されるようになってきたからである．画像も写真と見まちがうほどリアルになり，さらには車をねじったりする変形など，現実の世界ではありえないことも実現できる．コンピュータグラフィックスは，今後もっと独特の映像表現ツールに成長していくであろう．

3次元コンピュータグラフィックスにおける画像制作の作業は，大きく2つに分けることができる．ひとつは，データを作成する作業で，モデリング・ソフトウェアを使用する．モデリングでは，3次元物体（オブジェクトと呼ばれる）の形状，色や質感，物体の配置を定義していく．もうひとつの作業はレンダリングと呼ばれ，モデリング作業で作成されたデータを受け取り，画像計算を行う．本稿では，レンダリング技術について簡単に説明し，その中でもリアルな画像を計算できるレイトレーシング法に焦点をあてて解説していく．

レンダリング技術

レンダリングには，いろいろな手法がある．最近のグラフィックスワークステーションで採用されているグラフィックス専用チップは，Zバッファ法と呼ばれるレンダリング手法をハードウェア化している．これは，ポリゴンと呼ばれる多角形で構成される物体の奥行き（Z軸）座標をZバッファというメモリ（配列）に保存し，新たなポリゴンを描画する際に，このポリゴンのZ座標とZバッファの値を比較して，値が小さいとき（視点に近い）のみ描画を実行する，という方法である．

CGプロダクションで多く用いられているレンダリング手法は，スキャンライン法と呼ばれる．この方法も，Zバッファ法と同様にポリゴンで構成される物体を描画する．スキャンライン法は，Zバッファ法よりも計算量が多く，プログラミングも難しい．したがって，ハードウェア化が難しいのだが，よりリアルな画像が生成できる．

今回解説するレイトレーシング法は，光学に基づいて計算するため，Zバッファ法やスキャンライン法では表現できなかった鏡の映り込みやガラスの屈折（背景が歪む）が簡単に計算できる．また，ほかの物体に影を落とすこともできるので，超リアルな画像表現ができる．プログラミングにおいても，高校の数学程度の知識で簡単に書くことができることと，メモリをあまり必要としないので，パーソナルコンピュータでも十分使用できる利点がある．しかし，Zバッファ法やスキャンライン法にくらべると，計算時間がかかる欠点があるので，CGプロダクション会社ではあまり用いていない．

リアルな画像を生成できるもうひとつのレンダリング法にラジオシティ法がある．ラジオシティ法は，熱工学に基づいており，光源から放たれる熱がどのように伝達され，分布されていくかを計算し，その熱量を光量に置き換える方法である．レイトレーシングよりも照明シミュレーションにおいて正確な計算ができるため，微妙な影の表現ができるのが特徴である．

レイトレーシングの理論

レイトレーシング（ray tracing）とは，「光線を追跡する」という意味である．われわれが，「物が見える」と知覚するのは，光が物体に当たり，表面で光が一部吸収され，その残りが反射されて，その反射光が目に入るからである．レイトレーシングでは，この考え方をアルゴリズム化しようとしているのであるが，光源から放たれる光線を追いかけるには，次のような問題点があるため現実的ではない．

- 無数の光線を追いかけることになるため，膨大な計算量になる．つまり，いつまでたっても計算が終わらない．
- ほとんどの光線は目にはたどりつかない．目にたどりつく光線のみが視覚情報として有効であるから，無駄

グラフィックス用語集
スーパーインポーズ　コンピュータの画面をテレビやVTRの画面と重ね合わせること．単にコンピュータのテキスト画面とグラフィックス画面を同時表示する場合にも使われる．

が多い．

そこで，視覚情報として必要な光線のみを追いかけるように工夫をするため，目から光源へ逆に光線を追跡して計算する．図1は，この逆光線追跡を示したものである．レイトレーシング法は，厳密には逆光線追跡法であるのだが，一般には光線追跡法と呼ばれている．

レイトレーシング法を説明する前に3次元コンピュータグラフィックスの基本概念を復習しておこう．3次元コンピュータグラフィックスでは，カメラ，物体，照明をデータ定義しなければならない．カメラとは，視点（目の位置）とスクリーンと呼ばれる格子状の窓で構成される．スクリーンは，画像を計算する際の解像度を決定する．一般にはフレームバッファの解像度がスクリーンの解像度となる．

また，レイトレーシングで定義されるカメラは，ピンホールカメラと呼ばれる原始的なカメラに似ており，レンズをもたない．したがって，レンズによるさまざまな特殊効果や歪みは起こらない．

レイトレーシングで扱う物体は，先に説明したポリゴン形式の物体ではなく，プリミティブと呼ばれる数式（2次関数）で定義される数学的物体である．もちろん，ポリゴン形式の物体をレイトレーシングで計算することはできるが，応用編になるので今回の説明では省略する．

照明とは，光源（光の位置）とその種類で定義される．照明には，平行光源と点光源の2種類がある．平行光源とは，太陽のように無限の距離に位置する光源のことである．点光源は，豆電球のような小さい光源で，これに方向性をもたせた光源をスポットライトと呼ぶ．蛍光灯のような線光源や蛍光灯を数本並べた面光源は，直接には定義できないので，点光源を複数配置して定義する．

では，レイトレーシング法の基本概念を説明しよう．レイトレーシング法の計算方法は次の通りである．

①視点からスクリーンの1点（ピクセル）を通る方向に光線を逆にたどる．
②光線が物体に当たるまで追い続ける．
③物体に当たらなければ，①に戻り，スクリーンの次の点について調べる．物体に当たれば，
④物体との交点において，
1.物体が鏡である場合，光線の反射計算を行い，光線の方向を再定し，②に戻る．
2.物体がガラスである場合，光線の屈折計算を行い，光線の方向を再定し，②に戻る．
3.物体が上記以外である場合，その表面における質感を考慮した輝度計算を行い，その結果を画像データとする．①に戻り，スクリーンの次の点について調べる．

図1
逆光線追跡

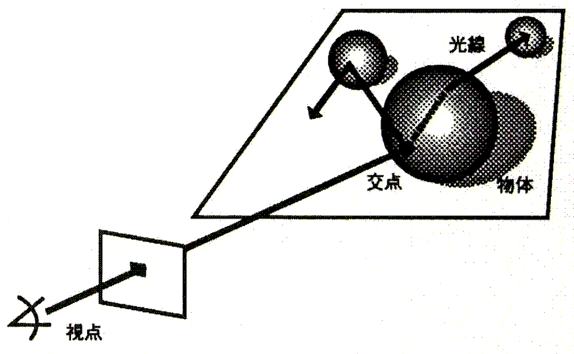
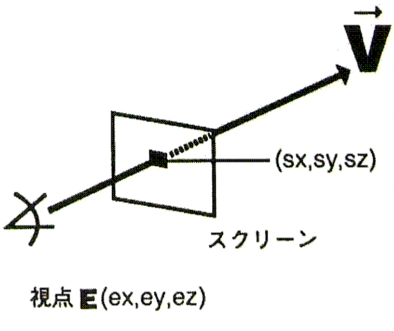


図2
視線ベクトル



グラフィックス用語集
スクロール　ディスプレイ上に表示されている画像を上下方向または左右にずらし，かくれて見えない部分をを表示させる方法．

視線ベクトル

視点からスクリーンの１点を通り伸びていく線を視線といい，プログラムでは視線ベクトルとして扱う（図２）．視線ベクトルＶ（vx，vy，vz）は，視点とスクリーン上の１点で結ばれるベクトルであるから，式１～３で表せる．

ただし，（sx，sy，sz）はスクリーン上の点，（ex，ey，ez）は視点である．CGでは，ベクトルは方向を定義するために用いられ，その大きさは重要でないことが多い．視線ベクトルも見る方向を意味しており，大きさは関係ない．したがって，ベクトル計算の都合上，大きさ（長さ）が１になるベクトルに統一しなければならない．これをベクトルの正規化という．視線ベクトルを正規化すると式４～７になる．

このベクトルの方向に視線ベクトルを伸ばすには，パラメトリックな式で表すと式８～10のようになる．

ｘ，ｙ，ｚは，視線ベクトルが視点からｔの距離だけ離れた空間を指す座標である．

式１・２・３

vx＝sx－ex……式１

vy＝sy－ey……式２

vz＝sz－ez……式３

式４・５・６・７

$q=\sqrt{(sx-ex)^2+(sy-ey)^2+(sz-ez)^2}$……式４

vx＝(sx－ex)/q ……式５

vy＝(sy－ey)/q ……式６

vz＝(sz－ez)/q ……式７

式８・９・10

x＝vx ＊ t＋ex……式８

y＝vy ＊ t＋ey……式９

z＝vz ＊ t＋ez……式10

プリミティブ

レイトレーシング法で扱う物体は，基本的にはプリミティブで定義される．プリミティブとは，関数（数式）で表される物体で，計算が簡単な２次曲面や平面および直方体を用いる．２次曲面は，球，楕円体，円錐，円筒などである．これらのプリミティブの方程式は，図３に示すとおりである．複雑な形状を作成するときには，これらのプリミティブを積木のように組み合わせる．

レイトレーシングでは，プリミティブの組み合わせをブーリアン演算（論理演算）と呼ばれるプリミティブ同士の組み合わせ計算で行う．ブーリアン演算には，論理和，論理差，論理積がある．図４は，ブーリアン演算の例である．論理差は，物体Ａから物体Ｂをくりぬき，論理積ではくりぬかれる部分のみがブーリアン演算の結果として残る．

図3
プリミティブの方程式

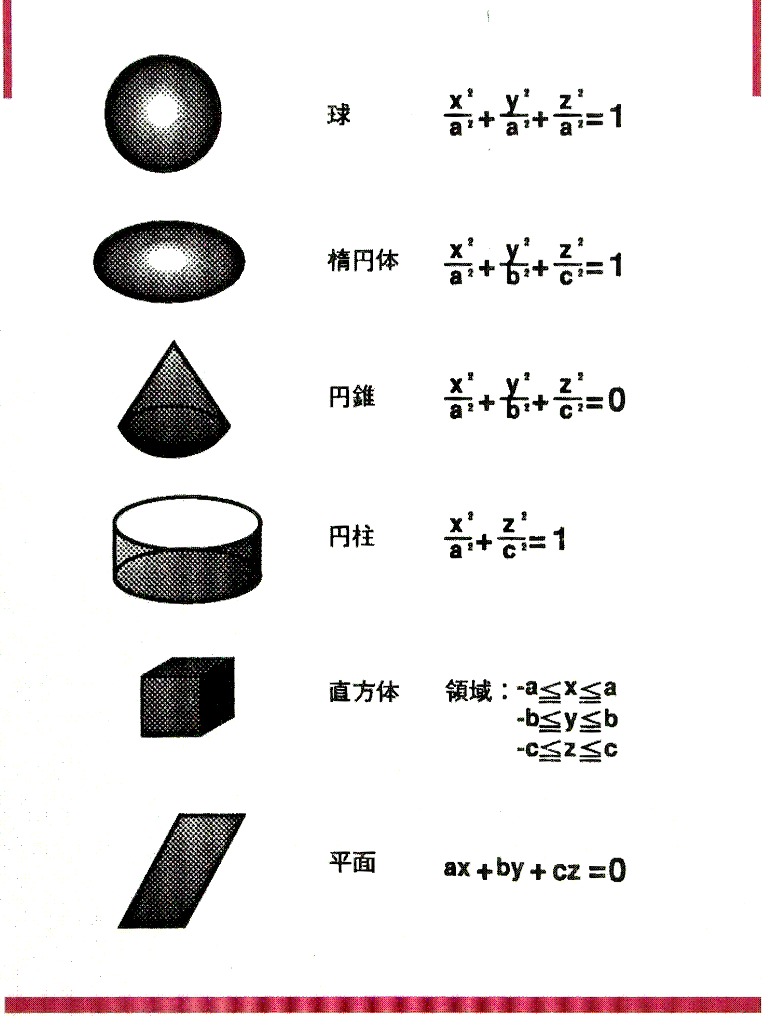
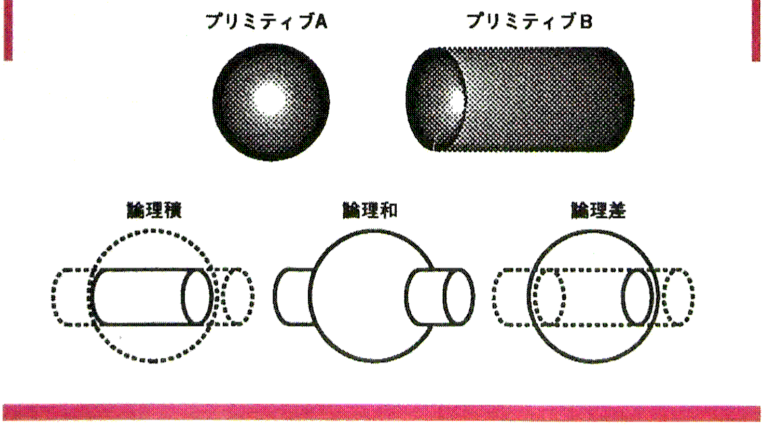


図4
ブーリアン演算の例



交点計算

プリミティブで定義されている物体と視線ベクトルの交点計算の方法について説明する．ここでは，プリミティブの例として球を用いて説明する．

球の方程式を式11に示す．

ただし，R：球の半径，(x0，y0，z0)：球の中心である．すなわち，この球の表面の点(x，y，z)は，この式を満たす．

式8，9，10と式11を比較してみよう．それぞれに座標(x，y，z)が登場する．この点こそ視線ベクトルと球との交点なのである．つまり，空間の同じ点を意味しているのである．したがって，式8，9，10を式11に代入できる．

式12において，唯一値が定まっていない変数は距離ｔである．ｔについて式12を整理してみると，式13のようになる．

この式13は，変数ｔの２次方程式 $at^2+bt+c=0$ である．式13に解があるかどうかは，判別式 $D=b^2-4ac$ を用いて調べることができる．判別式は次のように解釈する．

$D<0$　解なし：交点がない（視線ベクトルが球に当たらない）

$D=0$　解１個：交点は１点

$D>0$　解２個：交点は２点（球に入り，後から出る）

判別式 $D \geqq 0$ のときのみ，交点座標を計算する．交点座標を求める前に，まず式14を用いてｔを求める．

グラフィックス用語集
スプライト　アニメーションに使われるセルと同じ要領で，優先順位にしたがって複数の画面を重ね合わせる機能．セル単位で高速にスクロールさせたりできる．

式11

$$(x-x0)^2+(y-y0)^2+(z-z0)^2=R^2$$

式12

$$((vx * t+ex)-x0)^2+((vy * t+ey)-y0)^2+((vz * t+ez)-z0)^2=R$$

式13

$$(vx^2+vy^2+vz^2)t^2+$$
$$(2vx * ex-2x0 * vx+2vy * ey-2y0 * vy+2vz * ez-2z0 * vz)t+$$
$$(ex+ey+ez+x0+y0+z0-2x0 * ex-2y0 * ey-2z0 * ez-R^2)$$
$$=0$$

式14

$$t=\frac{-b-\sqrt{b^2-4ac}}{2ac}$$

D＞0の場合，交点が2点あるのだが，tが小さい値を有効とする．なぜならば，tは，視点からの距離を意味しているので，視線ベクトルが球に最初に当たるのは，tが小さいときである，tの値が大きいほうの解は，球の後ろ側での交点を意味しており，視点からは見えないので関係ない．式14で求められたtの値を式8，9，10に代入すると，求めた交点座標x，y，zが計算できるわけである．

反射と屈折

前項で計算した交点をもつプリミティブが鏡やガラスである場合，それぞれ反射，屈折の計算を行わなければならない．この計算は，視線ベクトルの方向を変えることが目的で，新たな視線ベクトルを用いて物体との交点を求めていく．

反射は，光学における光線の正反射の法則を用いる．図5に示すように視線ベクトルが物体の法線ベクトルに対して角度θで当たる．これを入射角というが，このベクトルは鏡面で反射されていく．反射される角度ηを反射角といい，入射角と等しい．新たな視線ベクトル(vx'，vy'，vz')は，式15～17で計算できる．

ただし，nx，ny，nzは，交点（x，y，z）における物体の法線ベクトルである．球の法線ベクトルは，式18～21となる．

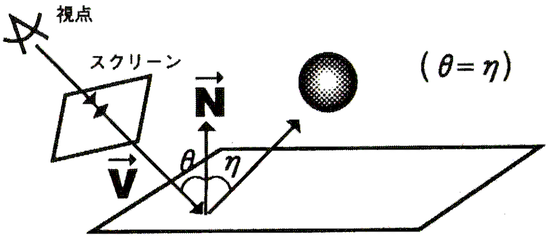
また，cosθはabs（vx＊nx+vy＊ny+vz＊nz）である．

屈折による新たな視線ベクトルは，反射と同様に求められる．屈折の場合，スネルの法則を用いる．屈折は，式

22～24で計算できる．

ただし，nは屈折率であり，空気の場合n＝1であるのに対して，ガラスはn＝1.5である．

図5



式15・16・17

$$vx'=vx+2\cos\theta * nx \quad \cdots\cdots 式15$$

$$vy'=vy+2\cos\theta * ny \quad \cdots\cdots 式16$$

$$vz'=vz+2\cos\theta * nz \quad \cdots\cdots 式17$$

式18・19・20・21

$$q=\sqrt{(x-x0)^2+(y-y0)^2+(z-z0)^2} \quad \cdots\cdots 式18$$

$$nx=(x-x0)/q \quad \cdots\cdots 式19$$

$$ny=(y-y0)/q \quad \cdots\cdots 式20$$

$$nz=(z-z0)/q \quad \cdots\cdots 式21$$

式22・23・24

$$vx'=\frac{1}{n}(vx-nx * (\sqrt{n^2-\sin^2\theta}-\cos\theta)) \quad \cdots\cdots 式22$$

$$vy'=\frac{1}{n}(vy-ny * (\sqrt{n^2-\sin^2\theta}-\cos\theta)) \quad \cdots\cdots 式23$$

$$vz'=\frac{1}{n}(vz-nz * (\sqrt{n^2-\sin^2\theta}-\cos\theta)) \quad \cdots\cdots 式24$$

輝度計算

プリミティブの交点における色を決定するために，輝度計算を行う．輝度計算とは，物体に光線が当たり，その光線が視線ベクトル方向にどのくらい反射するかを計算することを目的としている．つまり，交点における物体の明るさを計算するわけである．輝度は，光学の原理を簡略化して3つの要素に分けて考える．

3つの要素とは，間接光（アンビエント），乱反射光(ディフューズ)，鏡面反射光（スペキュラー）と呼ばれるハイライト光である．ただし，間接光は定数として扱い，計算を行わない．なぜならば，間接光は輝度の全体量のうちに占める割合が低いのであまり問題とならないし，実際に計算をすると膨大な計算量になってしまうからである．したがって，実際に計算する部分は乱反射光と鏡面反射光（ハイライト）である．

乱反射光とは，光線の面への入射に対して光線を無方向性反射することである．つまり，一方向に強く反射せずに，全方向に均一に反射するのである．この乱反射を簡略化した方程式をランバートの法則（Lambert's Law）という．乱反射は視点の位置とは関係がなく，物体と光源の相対的関係より算出できる．図6は，ランバートの法則を表している．ランバートの法則を用いて，乱反射光を式25で求める．

ただし，lx，ly，lzは，光源から放たれた光線が交点に到達する方向を意味する光線ベクトルであり，式26～29である．(jx，jy，jz)は点光源の位置である．

鏡面反射の場合，計算は少し複雑になるが，前項で説明した反射ベクトルの計算に似ている．鏡面反射は，光線

グラフィックス用語集
Zバッファアルゴリズム　3次元座標のZ軸方向への深さを基準として行うレンダリング法

の反射に方向性があるため，光源と物体の関係のほかに視点の位置も考慮に入れなければならない．ハイライトは，光線エネルギーを凝縮したスポットである．ハイライトは，鏡面反射で求めた方向性をもった光線強度をさらに強調する（式30）．

ただし，ax，ay，azは，式15，16，17で計算したvx'，vy'，vz'である．また，α はハイライト係数で，α が大きいほどハイライトはしぼり込まれ，金属的なシャープな感じになる．

このようにして，物体の表面はアンビエント光，ディフューズ光，スペキュラー光に分けられ，これらの成分の比率を調整することにより，つやをもつ質感，つやのない質感を表現する．

最終的には，赤，緑，青（RGB）の三原色に基づく加法混色の原理で定義される物体の色を考慮し，最終的な交点の色を決定する．この交点の色がスクリーンにおける色であり，すなわち画面上の1点の色となる．最終的な色（r，g，b）は，式31〜33のようになる．

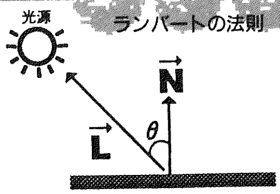
ただし，R，G，Bは物体色，SR，SG，SBは光源の色である．鏡面反射の輝度siは，光源の映り込みとして考えるため光源の色に依存している．

レイトレーシングの応用と最新動向

レイトレーシング法にさまざまな改良を加えることにより，よりリアルな画像を生成することができるようになったり，あるいはユニークな画像が制

図6

ランバートの法則



式25

$$di = lx * nx + ly * ny + lz * nz$$

式26・27・28・29

$$q = \sqrt{(jx-x)^2 + (jy-y)^2 + (jz-z)^2} \quad \cdots\cdots 式26$$

$$lx = (jx-x)/q \quad \cdots\cdots 式27$$

$$ly = (jy-y)/q \quad \cdots\cdots 式28$$

$$lz = (jz-z)/q \quad \cdots\cdots 式29$$

式30

$$si = (ly * ax + ly * ay + lz * az)^{\alpha}$$

式31・32・33

$$r = R * (ai + di) + SR \cdot si \quad \cdots\cdots 式31$$

$$g = G * (ai + di) + SG \cdot si \quad \cdots\cdots 式32$$

$$b = B * (ai + di) + SB \cdot si \quad \cdots\cdots 式33$$

グラフィックス用語集

ソリッドモデル　サーフェイスモデルに加え，中身の詰まった状態として表現する方法．切断面などの表現もできる．

作できるようになる.

パーティカル法

ユニークな画像生成を行う改良のひとつとして，パーティカル法がある．この方法は，粒子状の物体を表現することができる．粒状性物体は，微粒子の集団であり，式34～36で定義できる．

ただし，

a, b, c, d：パラメータ

x0, y0, z0：粒状性物体の中心

R：粒状性物体の大きさ

r1：粒状性物体の最低の大きさ

r2：乱数分布で決定される粒状性物体の大きさ

である．この式からわかるように，粒状性物体は球であり，この球の中に微粒子が無数に含まれているのである．

前項で球と視線ベクトルの交点を求める方法を説明したが，この交点を求める前に乱数を用いて粒状性物体の大きさ（半径）を設定する．つまり，粒状性物体の半径は，交点計算を行うたびに算出されるので，粒子のような球の断片が生成されることになる．写真1は，パーティカル法を用いた作品例である．

式34・35・36

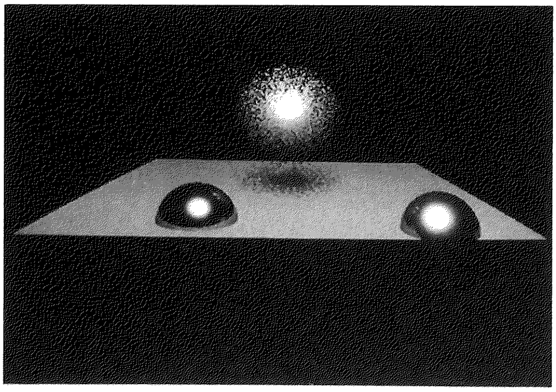
$$R^2=a(x-x0)^2+b(y-y0)^2+c(z-z0)^2 \quad \cdots\cdots 式34$$

$$R=r1+r2 \quad \cdots\cdots 式35$$

$$r2=d(random()) \quad \cdots\cdots 式36$$

r1＝定数

写真1　パーティカル法を用いた作品



相互反射

レイトレーシングの欠点のひとつは，環境光（アンビエント）を一定値に設定していることである．環境光は，光源から放たれる光線が壁や天井といったほかの物体に当たり，その物体面から反射して計算しようとしている物体面を照らすような光を表現するために用いられている．つまり，反射光源からの光が当たらない部分が真黒にならないようにする効果がある．このような2次反射（相互反射）を計算していないレイトレーシング法では，室内における照明効果を正確にはシミュレーションできない．この相互反射の問題を解決したラジオシティ法の概念をレイトレーシング法に取り入れようとする研究が行われている．

分散型レイトレーシング

分散型レイトレーシングでは，視線ベクトルや光線ベクトルを算出するとき，確率的に（つまり乱数を用いる）そのベクトルに対して少し方向をずらしたベクトルを16本程度作る．そしてこの16本のベクトルを用いて前項で説明した計算を行う．そして，最後の色決定のときに，16本の輝度の平均値を使用する．

分散型レイトレーシングでは，動きのブレや半影という輪郭がぼけた影を計算できる．しかし，通常のレイトレーシングよりも16倍以上も計算が遅くなるので，パーソナルコンピュータでは現実的ではない．

無限反射による作品例

無限反射は，2枚の鏡を平行に並べたときに起こる反射の繰り返しのことである．レイトレーシングは，反射や屈折の表現が得意であるから，無限反

グラフィックス用語集
タートルグラフィックス　画面上のタートルにコマンドを与え画面上にグラフィックスを描く．タートルには相対的な指定位置などのコマンドを与える．

図7

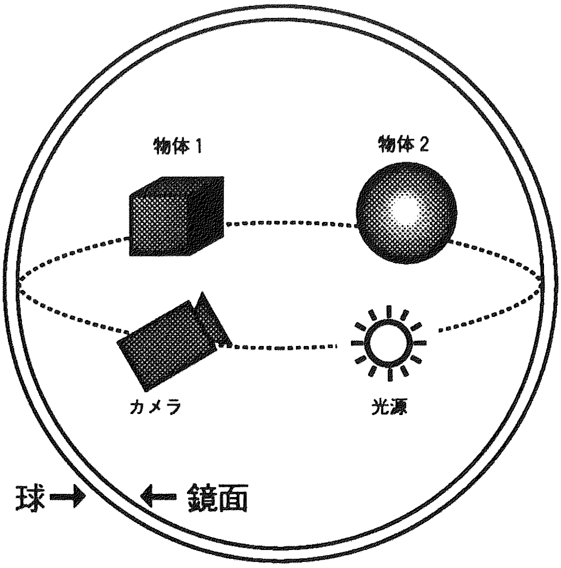


写真2

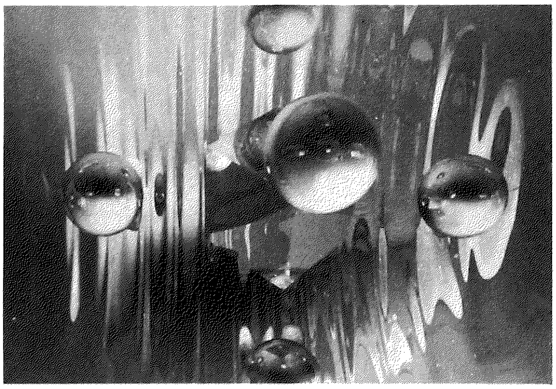
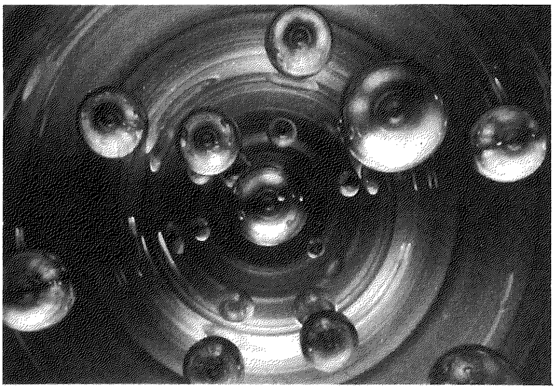


写真3



射となるようなデータを設定して計算させることにより，レイトレーシングでおもしろい画像が作れる．たとえば，図7のように大きい球の中を鏡張りにしてくりぬき，その中に物体，光源，視点を設定する．そうすると，交点計算で物体に当たらなくても大きい球の鏡面には必ず当たる．もし，物体にも鏡面を定義していたら，何回も映り込んで，非常に幻想的な画像が作成できる．

レイトレーシングは，超リアルな画像が計算できる，CGならではの手法である．手ではかけないような世界を計算によって創り出せる．しかし，今回紹介した方法は，あくまで基本部分であり，これに独自の方法を加えていくことにより，ユニークな画像を創りだせるのである．

参考文献

・Cook,R.L.,Porter,T.,and Carpenter,L.: "Distributed Ray Tracing," Computer Graphics,18,3,1984,pp.137-145

・Inakage,M.:"PARTICALS:An Artistic Approach to Fuzzy Objects," proceedings of CG International '88, pp.126-134

・Whitted,T.: "An Improved Illumination Model for Shaded Display," Comm.of the ACM,23,6,1980,pp.343-349

グラフィックス用語集
タイリング　一定のカラーパターンを並べたブロックを用意し，これをタイルを並べるように配置する．これによって見かけ上の色数を増やすことができる．

GRAPHICS PROGRAMMING

特集
グラフィックス
プログラミング
の世界

DOS/Vで注目される世界標準機のグラフィックス

IBM PCのビデオシステム入門

柳澤健二

ハードウェア的に日本語表示機能をもたない機種で日本語の使用を可能にするDOS/Vの発表，日本アイ・ビー・エム社によるオープン・アーキテクチャ推進協議会（OADG）の設立など，最近IBM PCがとみに注目を集めている．日本を除く世界の標準的パソコンであるIBM PCのビデオシステムが，日本で標準的な位置を占めるPC-9801のそれと，まったく異なったものであることは意外に知られていない．
世界標準パソコンであるIBM PCに搭載されているビデオシステムの種類と特徴について解説するとともに，IBM PCにおけるグラフィックスプログラミングの基本的な考え方を紹介する．

DOS/V（IBM DOSバージョン4.0/V）の登場もあって，このところ日本でも注目度の上がってきたIBM PCですが，たとえばPC-9801でのグラフィックスプログラミングに慣れた人が，この“世界標準パソコン”上でのそれに挑戦しようとした場合，もしかしたらかなりのカルチャーショックに襲われるかもしれません．

「えっ，98なら簡単なあの機能がない，この機能がない．なんて不便な世界だ！」
と，不満のかたまりになってしまうかもしれないし，逆に，

「じーん……，なんてスマートで柔軟性に富んだシステムなんだ……」
てな具合に，目からウロコをパラパラ落としつつ感涙にむせんでしまう可能性もあるかもしれません（ちょっとオーバーかな）．

まあ，どんな感想を抱くかは読者の皆さん1人1人におまかせするとして，とにかくIBM PC系パソコンでグラフィックスプログラミングを行ううえで最低限必要になると思われる知識について，以下にまとめてみたいと思います．

IBM PCのビデオシステム

（いわずもがなの前置きですが，この記事では現在のAT互換機やPS/2も含めて“IBM PC”と総称しますので，念のため……）

IBM PCのビデオシステムの最大の特徴は，なんといっても，

・本体とは独立した「ビデオサブシステム」のかたちで取り扱われ，用途や技術の進歩に応じて「互換性を保ったまま，別のビデオサブシステムに差し替えることが可能」

という点にあります．ステレオコンポみたいに，ユーザーが好みの「ビデオサブシステム」を，パソコン本体につけたりはずしたりできるわけです．

そうすることによって，たとえばいままでより高解像度の表示ができるようになったり，より高速に表示できるようになったり，はたまた1台のパソコンに複数のディスプレイアダプタ＋複数のディスプレイを接続して，同時に別々の情報を表示させたり（プログラムのデバッグなどに有効）といったことが，かなり気軽にできるようになっています．

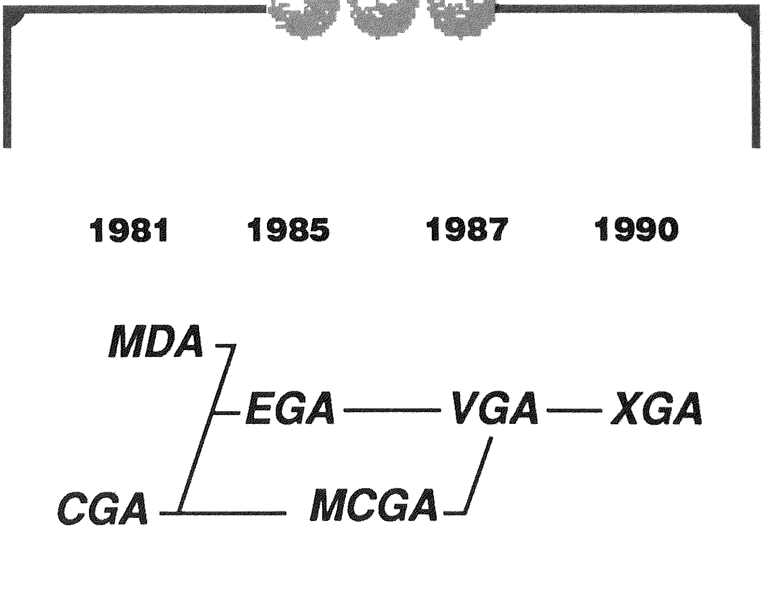
そういうわけで，プログラミングの実際についての話に入る前に，どうしてもまず「実際に，どのようなビデオサブシステムが存在するのか」という点に触れておく必要があるでしょう．

IBM自身が発表してきたビデオサブシステムは，図1のように，意外にシンプルな系統図（実はシンプルにするために一部省略してありますが）にまとめることができます．

これら以外にサードパーティ製のディスプレイボードがいろいろ，というより「星の数ほど」あり，それぞれ独自の仕様で機能拡張をはかっている場合も多いのですが，当然，図1のどれかのシステムとの互換性をもっています．

MDAとCGAには，すぐあとで触れるように互換性がありませんが，そのほかは右から左に向かって「互換性を保ちつつ機能の向上がはかられている」というふうにお考えください．たとえばCGA対応のグラフィックスのソフトウェアは，EGA，MCGA，VGA，そしてXGAでも動作可能ですから，VGA搭載のPCでCGA対応のゲームソフトを動作させるといったことが基本的に可能です．

図1



MDAとCGA

どちらも，初代IBM PCの発売当初から存在したビデオサブシステムで，MDAはMonochrome Display Adapter，CGAはColor Graphics Adapterの略です．

MDAは，その名前の由来通りモノクロ表示専用のアダプタで，720×350ピクセルというかなり高い解像度が特長です．ただし，なんとグラフィックス表示の機能がまったくなく，テキストモードを使用するアプリケーション専用です．

つまり，この記事のテーマ“グラフィックスプログラミング”は“不可能”なのですが，一般のビジネス用途にかなり普及した規格ですので，そういうものがあることは頭に入れておく必要があるかもしれません．

一方のCGAは，「カラー表示」と「グラフィックス表示」が可能なビデオサブシステムです．その名の通り，テキスト画面では80桁×25行×16色（前景・背景とも）のカラー表示が可能なほか，

- **640×200ピクセル×２色**
- **320×200ピクセル×４色**

のグラフィックス表示をサポートしています．解像度が最高でも640×200ピクセルということは，日本語だったら16ドットの文字でも10行ちょっとしか表示できません．しかも，そのモードで使える色は２色のみです．

アルファベット圏の文字であれば一応25行の表示が可能とはいえ，少々見づらいことにはかわりがありません．

要するに誕生当初のIBM PCのユーザーは，

「解像度は高いけれど，モノクロでグラフィックス表示機能がない」
「カラーでグラフィックス表示も可能だが，解像度は低い」

という，いってみれば「どっちもどっち」の２つから究極の選択？ を余儀なくされていたわけです．

もっとも，実はこれらは１台のマシンで共存可能なように作られており，「スプレッドシートへのデータ入力はMDA，グラフの表示はCGA」みたいなことも，専用ディスプレイ２台を接続することで可能ではあります．

具体的には，MDAとCGAとでは，使用されるビデオバッファ（VRAM）のアドレスが重なり合わないようにできています．すなわち，図２のように住みわけが行われており，MDAのディスプレイに文字を表示する場合にはB000Hからはじまるバッファ，CGA側に表示させたいときにはB800H以降のバッファにデータが書き込まれることになります．

ちなみにCGAとはまったく別の手法で，MDAとの互換性を保ちつつグラフィックス表示を可能にしたディスプレイアダプタもサードパーティから発売されており，これまた一時期，一種の標準とみなされるほどに普及しました．Hercules社のグラフィックスカード（HGC）というのがそれです．

EGA

さて，意地悪な見方をすれば「２つあわせて一人前」みたいなところのあったMDAとCGAの２つを統合し，さらに表示性能を拡張したものが，1985年に登場したEGA（Enhanced Graphics Adapter）です．

“エンハンスト（拡張された）”という名前の通り，グラフィックス面ではCGAの上位互換という性格をもつ一方，MDA固有のモノクロテキスト画面にも対応しています．

つまり，EGAを搭載したIBM PCでは，CGA対応のソフトもMDA対応のソフトもそのまま動作するうえ，

(1)320×200ピクセル×16色
(2)640×200ピクセル×16色
(3)640×350ピクセル(モノクロ)
(4)640×350ピクセル×16色
(いずれも64色中)

……など，EGA固有の新しいグラフ

図２

```
MDA    B000H:0000H－B000H:07FFH
CGA    B800H:0000H－B800H:4000H
```


グラフィックス用語集
テクスチュア　質感のこと．画像に陰影を付ける際に，コンピュータに与える情報には光源の方向・明るさおよび面のテクスチュアの３つが必要となる．

ィックスのモードがそこにつけ加えられるしかけです.

このうちもっとも解像度の高い(4)の画面モードは，たとえばゲームソフトなどでしばしばお目にかかるものです.

これら4つの新しいグラフィックスのモードでは，使用されるビデオバッファのアドレスがMDAともCGAとも異なり，セグメントA000Hからはじまります.一方，MDAやCGAのモードをエミュレートする際には，それぞれのオリジナルと同様，B000HまたはB800Hが使用されます.

ちなみに，普通のテレビと同様のコンポジットビデオ信号を用いていた（したがって普通のテレビにも映る）CGAに対して，EGAではRGBモニタを必要とします.

MCGAとVGA

1987年，IBM PS/2に搭載されてデビューしたのが，VGA（Video Graphics Array）とMCGA（Multi Color Graphics Array）です.

もちろん，従来のIBM PC/AT用のVGAアダプタもIBM自身やほかの周辺機器メーカーなどから発売されており，それまでEGAなどを搭載していたマシンにセットすることで，VGA対応パソコンにグレードアップすることが可能です.

VGAの基本コンセプトは，EGAと同様，

「従来のおもなIBM純正ビデオグラフィックス規格をすべて包含し，さらにそれらを拡張したもの」

といえます.つまり，直接的にはEGAに対して上位互換となっており，したがってMDAやCGAに対しても上位互換であるほか，

(1)320×200ピクセル×256色
(2)640×480ピクセル×2色
(3)640×480ピクセル×16色
（いずれも262144色中）

などの新しい画面モードを備えています.なお，ディスプレイにはアナログRGBが必須です.

いまではすっかり主流の規格となっているこのVGA，日本アイ・ビー・エム社が昨年発売した「DOS/V」が，これを前提にしていることはすでにご存じの方も多いことでしょう.

なお，MCGAはPS/2モデル25，30などの“下位機種”に搭載されたもので，いってみればVGAのサブセットのような性格のものです.CGAに対しては互換性が保たれていますが，MDAやEGAとの互換性はありません.

すでにVGA全盛のいま，あまり実物にお目にかかる機会もなさそうですが，(1)(2)のモードはこちらにも用意されており，256色表示対応のカラフルなゲームソフトなどはこのMCGAでも動作します.

グラフィックスプログラミングの実際

以上，MDAから今主流のVGAまで大ざっぱに概観したところで，それらのビデオサブシステム上でのグラフィックスプログラミングについて説明します.

表1は，それぞれのビデオサブシステムがサポートしているビデオモードをまとめたものです.

左端の「ALの値」という見出しをつけた項目に入っている番号は，各ビデオモード固有のいわば“背番号”です.

たとえば，CGAでは背番号00Hから06Hの7つ，MDAでは07Hだけ，EGAではそれらに加えて0DHから10Hの4つ，VGAではさらに11Hから13Hまで3つの背番号（うち2つはMCGAと共通）が加わっていることが見てとれるでしょう.

野球チームでいえば，古参選手が1人も引退しないまま新人選手がどんどん加わって，だいぶ大所帯になってきたようなイメージです.

したがって，最大公約数的な存在になっているCGA用に作られたグラフィックスのプログラムは，MDAを除くどのアダプタを搭載したマシンでも動作します.

もっともこの場合には，EGAやVGAのもつ本来の性能は活かせないことになりますが，よく見ると04H，05H，0DH，13Hの4つは，解像度が320×200ピクセルでまったく同じであり，違うのは使える色数だけです.つまり，4色までしか表示できないCGA用に開発されたソフトを，256色同時発色可能なカラフルな環境に移植することも，それほどむずかしくないのです.

グラフィックス用語集
デュアルポートRAM　グラフィックスのディスプレイ画面への表示を，スピーディに，かつ効率的に行うために，書き込みと読み出しが同時にできるメモリ.

ROM BIOSのビデオサービスルーチン

IBM PCのBIOSには，画面関係を操作するプログラミングのために有用な機能が「INT 10H」のサービスの中にまとめられており，アセンブリ言語をはじめ，割り込み手続きをサポートしているCやPascal(たとえばTurbo Pascal)，BASIC(たとえばQuickBasic)などの言語からも(必要ならば)直接呼び出すことができます．

具体的には，個々のサービスごとに必要なパラメータなどを各レジスタにセットし，AHレジスタに機能番号をセットして，INT 10Hを発行するかたちとなります(表2参照)．

たとえば，ディスプレイをCGAの320×200×4色グラフィックスのモードにセットする場合でいうと，

- **「スクリーンモードの設定」の機能番号は00H．**
- **設定したい画面番号は04Hで，これはALに指定する決まりになっている．**

……わけですから，実際の作業をアセンブラの形式で表現すれば，図3のようにすればよいことになります．簡単ですね．

ちなみに，PC-9801シリーズのグラフィックスプログラミングに慣れた方などは，多少とまどうかもしれないのですが，実はIBM PCでは，文字表示関連のサービスはグラフィックス画面でもテキスト画面とほぼ同様に利用できます．

つまり，現在の画面がどちらのモードであるかに関係なく，任意の位置に文字を表示させたり画面の一部をスクロールさせたり，さらには画面上の任意の位置に表示されている(はずの)文字のキャラクタコードを読み取ったりすることが可能なのです．

その代わりというのもなんですが，グラフィックス画面とテキスト画面の重ね合わせ，といった機能は，どのビデオサブシステムでもサポートされていません．

なお，これらのサービスルーチン自体，MDA／CGAからEGA，VGAに向かって「上位互換」の方式がとられており，最初からシステムボードに搭載されているのはMDAとCGAレベルのものに限られます．

では，ディスプレイアダプタをEGAやVGAなどに差し替えた場合はどうなるかというと，そうした新しい表示規格のための拡張機能は，ディ

表1　おもな純正ビデオサブシステムのビデオモード番号と仕様

ALの値	タイプ	文字	ピクセル数	1ピクセルビット数	パレット色数	表示色数	バッファ開始SEG	MDA	CGA	EGA	MCGA	VGA	DOS/V
00H	テキスト	40　25				16	B800		○	○	○	○	△※2
01H	テキスト	40　25				16	B800		○	○	○	○	△
02H	テキスト	80　25				16	B800		○	○	○	○	△
03H	テキスト	80　25				16	B800※1		○	○	○	○	○
04H	グラフィック	40　25	320×200	2	16	4	B800		○	○	○	○	△
05H	グラフィック	40　25	320×200	2	16	4	B800		○	○	○	○	△
06H	グラフィック	80　25	640×200	1	16	2	B800		○	○	○	○	△
07H	テキスト	80　25				MONO	B000	○		○		○	△
0DH	グラフィック	40　25	320×200	4	64	16	A000			○		○	△
0EH	グラフィック	80　25	640×200	4	64	16	A000			○		○	△
0FH	グラフィック		640×350	1	64	2	A000			○		○	△
10H	グラフィック		640×350	4	64	4/16	A000			○		○	△
11H	グラフィック※3		640×480	1	256K	2	A000				○	○	○
12H	グラフィック※3		640×480	4	256K	16	A000					○	○
13H	グラフィック		320×200	8	256K	256	A000				○	○	○
72H	グラフィック	80　25	640×480	4	256K	16	----						○
73H	テキスト	80　25	640×480		256K	16	----						○

《注》
- AL =0,1/2,3/4,5 はそれぞれ同一のモードだが，偶数の値を設定した場合，以前のモードでVRAMに書き込まれたデータがクリアされずに残る．奇数の場合は画面が初期化される．
  テキスト画面の解像度はアダプタごとに異なる．
- 欠番になっているモード番号も未使用というわけではなく，内部的に使用されるもの，IBM製の一部機種で使用しているもの，サードパーティ製ビデオアダプタで拡張機能呼び出し用に使用しているものなどがある．

※1：DOS/Vではこのアドレスは使用されない．
※2：△は，DOS/V日本語環境においては日本語表示のサポートが行われず，動作の保証もされていないが，設定すること自体は可能．
※3：DOS/Vではこのモードで80×30行の日本語テキスト表示が可能．

グラフィックス用語集
ドット　グラフィックスを表示する最小構成単位のことで，これが多いほど細かい表示が可能になるが，その分メモリも必要になる．

スプレイアダプタ側にROMのかたちで搭載されており，これが本来のBIOSルーチンをまるごと代替する（INT 10Hを横取りする）かたちではたらくようになっています．

この仕組みによって，CGAでは利用できないさまざまな機能(たとえば，VGA固有のグラフィックス画面を呼び出す)が新たにつけ加わってくるわけで，結果，IBM PCのディスプレイまわりのBIOSは，十分とはいえないまでも，つねに必要最低限の機能を満たした存在となっています．

したがって，スピードに難はあるものの，画面表示を完全にBIOSに依存したかたちで，たいていのアプリケーションを開発することが論理的には可能です．

高速化をはかる場合にも，まずBIOSを使用してプログラムがきちんと動作するところまでもち込んだうえで，個々の機能を代替するルーチンを新たに作って差し替える，といったかたちがとれることになります．

とくにテキスト画面だけを使うプログラムの場合，実際，すべてBIOSに依存しているもの，あるいは動作環境設定メニューで「画面表示をVRAMを直接読み書きする高速ルーチンで行うか，BIOS経由で行うか」を選択できるものもしばしば見かけます．

こうした「行儀のよい」ソフトは，たとえば後述のDOS/Vの日本語モードのように，セグメントB800Hの位置にVRAMをもたないやや特殊な環境でも，問題なく画面表示が可能になります．

一方，グラフィックス画面を使用するプログラムの場合はどうかというと，BIOSがサポートする表示機能は残念ながら「1ドット単位の書き込みと読み込み」だけで，画面に線を引いたり円や矩形を描いたり塗りつぶしたり，といったところまではめんどうをみてくれません．しかしながら，スクリーンモードのセット，パレットの変更など，補助的な操作に関するサービスは十分に利用価値の高いものです．

もっとも，グラフィックス表示全般の機能を備えている(もしくはライブラリのかたちで提供されている)プログラミング言語を使用する場合には，ことグラフィックスプログラミングに関するかぎり，プログラマがこれらのBIOSルーチンを直接呼び出す必要はあまり多くないかもしれません．

プログラミングの流れ

どんなプログラミング言語を使うにせよ，IBM PC上でのグラフィックスプログラミングの全体の枠組みは，だいたい次のようになります．

(1)ディスプレイアダプタの種別の確認
(2)スクリーンモードの設定
(3)パレットの設定
(4)以下，ドットの読み書きなど，プログラム個々の実際の動作
(5)スクリーンモードをオリジナルの状態（通常はテキストモード）に戻して終了

……いくつか補足しますと，まず(1)の「ディスプレイアダプタの種類の確認」は，さまざまな仕様のディスプレイアダプタが存在する以上，自分のマシン以外の環境で使われる可能性がある場合には，当然なんらかのかたちで必要になってきます．

これは，ユーザーの手をわずらわせずに自動的に設定できればそれに越したことはないわけですが，実際のところ「互換機の存在」「サードパーティ製ビデオアダプタの存在」「複数のアダプタが同居している可能性」などを考慮に入れつつ，自動的かつ正確にこの作業を行うのは至難のワザです．

表2で示した通り，PS/2以降のビデオBIOSには，接続されているビデオサブシステムやディスプレイのタイプを取得できる1AHのサービスが用意されていますが，EGA搭載のPC/ATなどでは使えませんし，PS/2に他社製のディスプレイアダプタを増設している場合などにも，正しい情報が取得できる保証はないことになってしまいます．

要するにIBMのビデオシステムの柔軟性が裏目に出て，接続されているビデオサブシステムやディスプレイ自体の種別を簡単に知る方法は実際のところ“ない”のです．

プログラムの目的にとっては本質的なことではないので，アマチュアプログラマの場合は必ずしもそこまでこだ

図3

```
mov     ax,     0004h    ;ah=00h ...機能番号
                         ;al=04h ...画面番号
int     10h
```

グラフィックス用語集
トランスピュータ　グラフィックス描画は解像度や色数が増えるにしたがってCPUにかかる負担も多くなる．トランスピュータは本体CPUと並列にこれらの処理を行う．

わることはないわけですが，最低限「このソフトはVGA専用」みたいな具合に，使用条件がユーザーにわかるようにしておく必要はあります．

市販アプリケーションでも，インストール時にユーザーが自分のシステム構成をアプリケーションに知らせることで，この確認作業をすませているケースが多いようです．

次に，(2)のスクリーンモードの設定と，(5)の「終了時に元のスクリーンモードに戻す」作業ですが，IBM PCのMS-DOSのデフォルト（起動時）の画面モードはつねにテキスト画面であるため，グラフィックスプログラミングではこれも必須ということになります．プログラム起動時のビデオモードはビデオBIOSのAH=0FHの機能でわかりますので，起動直後にまず画面モードを調べ，終了時には再びそのモードに戻しておくのが礼儀でしょう．

表2　INT10HのビデオBIOS

AHの値	機能内容	呼び出し時の設定	レジスタに返される値	備考
00H	ビデオモードの設定	AL＝ビデオモード番号	なし	
01H	カーソル形状の設定	CH＝カーソル上部の走査線番号 CL＝カーソル下部の走査線番号	なし	
02H	カーソル位置の設定	BH＝ビデオページ DH＝行（0〜） DL＝桁（0〜）	なし	
03H	カーソルの状態（位置と形状）の取得	BH＝ビデオページ番号	CH＝いちばん上の走査線番号 CL＝いちばん下の走査線番号	
04H	ライトペン位置の取得			
05H	ビデオページの選択	AL＝ビデオページ番号	なし	
06H	画面のスクロールアップ……画面の一部（ウィンドウ）単位でのスクロールが可能．	AL＝スクロール行数 BH＝新たなブランク行の表示属性 CH＝左上隅の行位置 CL＝左上隅の桁位置 DH＝右下隅の行位置 DL＝右下隅の桁位置 （BH＝0でウィンドウ内クリア）	なし	
07H	画面のスクロールダウン	06Hと同様	なし	
08H	カーソル位置の文字コードと表示属性値の取得	BH＝ビデオページ	AH＝属性値（テキスト画面のみ） AL＝文字コード	
09H	現在のカーソル位置への文字と属性の書き込み	AL＝文字コード BH＝ビデオページ BL＝属性値（テキスト） 　　前景色（グラフィック） CX＝繰り返す文字数	なし	
0AH	現在のカーソル位置への文字の書き込み	AL＝文字コード BH＝ビデオページ BL＝前景色（グラフィック） CX＝繰り返す文字数	なし	
0BH	CGA 4色パレット設定	BH＝0 　BL＝ふち画面色（テキスト） 　BL＝背景色（CGA320×200） BH＝1（CGA320×200） 　BL＝パレット番号	なし	
0CH	ピクセルの書き込み	AL＝色番号 BH＝ビデオページ DX＝Y座標（0〜） CX＝X座標（0〜）	なし	グラフィックのみ
0DH	ピクセルの読みとり	BH＝ビデオページ DX＝Y座標（0〜） CX＝X座標（0〜）	AL＝色番号	グラフィックのみ
0EH	テレタイプ式書き込み	AL＝文字コード BL＝前景色（グラフィック） BH＝画面ページ（一部機種）	なし	
0FH	ビデオモードの取得	なし	AL＝ビデオモード番号 AH＝1行の桁数 BH＝アクティブ画面のページ	

グラフィックス用語集
ハイライト効果　物体の鏡面反射により光源が間接的に視点に届く状態．物体と光源だけでなく視点を含めた位置関係に影響される．

スクリーンモードの設定は，前のほうでも触れたようにAH=00Hの機能を使って行えます．ただし，たとえばCGAしかのっていないマシンで，VGA固有のモードを呼び出しても，もちろんそのモードにはならないわけですから，正しく設定されたかどうか，もう一度AH=0FHでチェックしてみる必要があります(BIOS側にはこ

10H	カラーパレットインターフェース	ALにサブサービス番号を指定することにより，EGA以降のカラーパレット設定を制御する16の機能が利用できる． （例） AL＝10H ……VGAのパレット個別設定 BX＝変更するパレット番号 DH＝R（赤） CH＝G（緑） CL＝B（青） AL＝12H ……VGAのパレットを一括設定 ES：DX＝テーブル・アドレス BX＝先頭のパレット番号 CX＝設定するパレットの数		EGA以降
11H	キャラクタジェネレータ・インターフェース	ALにサブサービス番号を指定することにより，英数字キャラクタの読み込み，定義などが制御できる（詳細省略）		EGA以降
12H	ビデオサブシステムの各種設定の切り替え	BLにサブサービス番号を指定することにより，ビデオサブシステムの各種状態の読み取り，設定変更を行う．（詳細省略）		EGA以降
13H	文字列の一括読み込み，書き込み	ES：BP＝文字列のポインタ CX＝文字列の長さ DX＝書き込む位置 BH＝ビデオページ AL＝00H・01H BL＝属性 ……単一の属性で書き込み．01Hの場合，文字列末尾にカーソルが移動する AL＝02H・03H ……属性つき書き込み．1文字単位の属性情報つき文字列データを用意する必要がある．		初期のROM BIOSにはない．
1AH	ディスプレイ組み合わせコード（コンビネーションコード）読み取りと設定	AL＝00H ……接続されているビデオサブシステムとディスプレイの種別を得る． AL＝01H ……アクティブ／非アクティブの切り替えその他の設定変更	AL＝1AH（この機能がサポートされている場合） BL（アクティブ） BH（非アクティブ） ＝00H　ディスプレイなし ＝01H　MDA ＝02H　CGA ＝04H　カラーEGA ＝05H　モノクロEGA ＝06H　PGC ＝07H　モノクロVGA ＝08H　カラーVGA ＝0BH　モノクロMCGA ＝0CH　カラーMCGA AL＝1AH	PS/2またはVGA搭載のPCのみ
1BH	ビデオBIOSの機能／状態情報の取得	BX＝0 ES：DI＝情報取得用の64バイトのバッファのアドレス	AL＝1BH（機能がサポートされている場合） 指定したバッファに詳細な情報が返される．	PS/2またはVGA
1CH	ビデオ状態のセーブとリストア	（詳細省略） AL＝0　取得 AL＝1　保存 AL＝2　復帰		VGAのみ

グラフィックス用語集
バックグラウンドカラー　グラフィックス画面の背景の色のこと．画面を消去したときなどは，この色で埋められる．

うしたエラーチェックの機能はありません).

パレット

さて，(3)の「パレットの設定」です．

「パレット」という概念自体は，別にIBM PC独自のものではないので，いまさら説明するまでもないかもしれませんが，とにかくIBM PCのカラーグラフィックスの画面モードには，例外なくこの「パレット」の機能が存在し，グラフィックスのプログラムを作るさいには大きなポイントになります．

ビデオBIOSの0CH(ピクセルの書き込み)の項目を見ると，ALレジスタに1バイトの「色番号」を指定するようになっています．

たとえばCGAの2色のモードでは0番と1番，EGAの16色のモードでは0番から15番(0FH)，VGAの256色のモードでは0番から255番(FFH)までの色番号が指定できます．

この色番号というのは，0番は黒，1番は青……というようにあらかじめ特定の色が決まっているのではなく，あくまでも「0番の色」「1番の色」「2番の色」というような抽象的な背番号にすぎません．

ここで，その抽象的な色番号を，具体的な色に結びつけるための索引の役目を果たすのが「パレット」というわけです．

もっとも，自動的に設定されるデフォルトの色がありますので，それを使っているかぎりでは「色番号1は黒，2は青……」みたいな覚え方をしてしまっても結果は同じですが，実際にそれぞれの番号にどんな色を割り当てるかは，基本的にはプログラマの自由です．

このパレットの設定に使用できる色の種類は，ビデオサブシステムごとにさまざまですが，EGAでは64色，またMCGAとVGAでは256K(262144)色までのバリエーションの中から，スクリーンのモードしだいで2色，4色，16色，あるいは256色を使うことができます．

こんなふうに，実際に選べる色数に大きな違いがありますので，パレットの設定方法自体もビデオサブシステムによって異なっています．

まずCGA(320×200×4色のモード)では，AH=0BHの機能を用いて，2つある既成の4色パレット(緑，赤，黄／シアン，紫，白)のうちどちらか1つを選択できるだけです．

EGAでは，AH=10Hの機能を用いて，どのグラフィック画面のパレットも，1色ごとに64色中から選択できます．

VGAでも，やはりAH=10Hにまとめられている(VGA用に拡張された)機能を用いて，R，G，Bそれぞれ1バイトのデータの形で各色に微妙な色合いを与えることが可能です．ちなみに，このVGAの豊富な発色能力を支えている機構は，ビデオDAC(Digital Analog Converter)と呼ばれています．

なお，CGA，EGA，VGAは右に向かって上位互換にあるわけですから，たとえばVGA上で従来のEGAと同じやり方でパレットを設定することも可能です．

描画ルーチン

すでに触れた通り，IBMのビデオBIOSにおいて，実際に画面にグラフィックスの描画を行うために用意されているのは「画面の任意の位置への1ドット単位の書き込み／読み込み」の機能だけです．

つまり，画面に線を引いたり円を描いたり，決められた範囲を塗りつぶしたり……といった機能は，すべて個々のアプリケーションソフト側が自分で用意する(現実的には市販のグラフィックス対応のプログラミング言語，もしくはライブラリなどを利用する)ことになります．

したがって，実際のグラフィックスプログラミングのテクニックは個々の言語やライブラリの機能や仕様しだいということになりますので，ここでは省略させていただきます．

どのみち，各種のビデオサブシステムがサポートするグラフィックス機能の共通点と特性とを一通り把握してしまえば，あとの実際のプログラミングの作業は，IBM PCと互換性のない他のMS-DOSパソコンでのそれと，あまり変わらないことも確かです．

スーパーVGAとXGA

以上，VGAまでのビデオサブシステム上でのグラフィックスプログラミングについて紹介してきましたが，最近の本家IBM PCの世界では，VGA

グラフィックス用語集
パレット　少ないメモリを効率良く発色するために16色などのテーブルを用意し，発色可能な色から任意の色をテーブルに割り当てる．このカラーテーブルをパレットという．

よりさらに上をゆく「スーパーVGA」や「XGA」といったグラフィックスアダプタの話題がよく聞かれます．これらは，いったいどういうものなのでしょうか．

「スーパーVGA」というのは，実際にそういう名前のモノ，あるいは仕様が存在するわけではなく，サードパーティが開発・販売しているVGA“上位互換”のアダプタを総称していう言葉です．

たとえば，640×480ピクセルのモードで256色が使えたり，800×600，あるいは1024×768ピクセル，といった具合にVGAより高解像度の表示ができるようになっていたり……など，その“上位”たるゆえんはさまざまです．もちろん，その一方ではIBM標準のVGAの機能も備えているわけで，ここではとりあえず「VGAの変種」ととらえ，あまり深く立ち入らないことにします．

IBM自身が発売しているVGAより高解像度のビデオサブシステムとしては，1987年に発表された「8514/A」と，昨年発表された「XGA(eXtended Graphics Array)」があります．XGAは，VGAと8514/Aの両方に対して上位互換性をもちます．

XGAは今後の普及が期待されている規格ですので，少しその仕様について触れておきます．

XGAは，VGA互換モードと拡張機能をもっており，後者では，

- **解像度**
  1024×768ピクセル／640×480ピクセル
- **色数**
  最高262144色中16色（VRAM512KBの場合）または256色（VRAM1MBの場合）．モノクロームの場合，16または64階調．

……という高い表示性能をもっています．高いといっても日本のPS/55の高解像度対応機種と同等(互換性はない)ですが，処理スピードが非常に高速であること，WYSIWYGの時代を睨んださまざまな拡張ファンクション（ライン，塗りつぶしなどの機能やスケーラブルフォントへの対応，スプライトの高速移動など）が用意されている，などの特徴をもっています．

すでに，同等の解像度を実現する各社の「スーパーVGA」がかなり安価に出回っている現在，この新たな規格がどこまで普及するのかは未知数ですが，今後，DOS/Vの兄貴分にあたる「DOS/X」が，このXGAを土台に発表されることなども期待され，むしろわれわれ日本人を含む2バイトコード系の複雑な字形を必要とする民族が，普及のカギを握っているといえるかもしれません．

DOS/Vのグラフィックスモード

最後に，話題のDOS/V(IBM DOS J4.0/V）についても触れておきます．

DOS/Vの特徴をおおざっぱにまとめると，次のようになるでしょう．

- **“VGA”という特定のビデオサブシステムを前提に，**
- **コンピュータ本体やVGAのハードウェアにはいっさい変更を加えず，**
- **英語版のMS-DOS（IBM DOS）の基本的部分もそのまま利用したうえで，**
- **デバイスドライバなどのソフトウェア上の工夫だけで日本語環境を実現した．**

本来，日本語表示を前提に設計されていないハードウェアで16ドット角の日本語フォントを表示させるため，「カラーテキスト画面」にあたるものを日本語環境ではVGAの16色グラフィックス画面を用いてエミュレートし，BIOSのレベルでは「テキスト画面」であるかのように扱えるようにしているわけです．

ちなみに，実は英語モードのVGAの標準的なテキスト画面(CGA互換のモード3H)は640×400ピクセルですが，日本語モードのテキスト画面は640×475ピクセルとなっています．

もっとも，テキスト画面の表示はキャラクタ単位で行われるわけで，実際の解像度がどうだろうとプログラマがその違いを意識する必要はあまりないのですが，画面のビデオバッファに直接読み書きを行う英語モードのアプリケーションに関しては，本来のカラーテキストモードが使用するB800Hからのバッファが，DOS/Vの日本語モードでは使用されていないために，「動作はしているけれども画面にまったく文字が表示されない」という現象が起きることになります．

ただし，BIOSを使用して文字表示を行っているソフトの場合は問題ありません．この点については，先に触れた通りです．

というわけで，テキスト画面に関しては本家の環境との間に多少のズレのあるDOS/Vですが，グラフィックスのプログラミングに関してはあまり問題はありません．

表3は，DOS/Vが「日本語環境」として設定しているビデオモードの一覧ですが，表1と比べてみると，11H(2

グラフィックス用語集
ピクセル　画素（picture element）の省略形．グラフィックスを表示する最小構成単位のことで，これが多いほど細かい表示が可能になるが，その分メモリも必要になる．

色グラフィックス）と12H（16色グラフィックス）の２つに関しては，本来のVGAがサポートしているのとモード番号，解像度，色数などがすべて同じであることがわかります（ちなみにこの２つのモードでは，日本語フォントが30行分表示でき，通常の25行モードより見通しがよいため，グラフィック系ソフトのみならず日本語ワープロソフト，あるいはエディタなどにもぜひ対応してほしいモードです．現在，一太郎 dashがこのモードを選択できるようになっています）．

これらのモードでは，ビデオバッファのアドレスなども英語環境とまったく同一で，ただ単に「BIOSを利用すれば日本語を表示したり，あるいはフォントパターンを読み出したりすることが可能」という違いしかないわけです．

このため，グラフィック画面を使用する英語モード用ソフトの多くは，DOS/Vの日本語環境の上でそのまま動作します．

正式にサポートされている11H，12Hのモードに限らず，たとえばCGA互換の04Hのモードなども使用することができます．ただし確実な動作が保証されているわけではないし，日本語の表示もできなくなります．

そんなわけで，プログラミングの側面からいうと，グラフィックスプログラムの基本部分は，本家IBM PC用プログラミング言語のVGA対応ルーチンを使用し，日本語の文字表示やフォントの読み出しの部分だけをBIOS経由で行えば，たいていの用はたりることになります．

要するにDOS/Vは，とくにグラフィックスプログラミングの環境に関するかぎり，日本と海外との壁を非常にスマートな手法でとっぱらってしまったのです．

参考資料

「IBM PC & PS/2 プログラマーズガイド」，翔泳社（The New Peter Norton Programmer's Guide To The IBM PC & PS/2 by Peter Norton & Richard Wilton）
「PC & PS/2 ビデオシステム プログラマーズガイド」，翔泳社（Programmer's Guide To PC & PS/2 VIDEO SYSTEMS by Richard Wilton）
「プログラマーのためのPCソースブック」，翔泳社（The Programmer's PC Sourcebook by Thom Hogan）
「IBM DOS J4.0/V BIOSインターフェース技術解説書」，日本アイ・ビー・エム
「IBM Personal System/2 XGA Adapter Interface Technical Reference」，International Business Machines Corporation

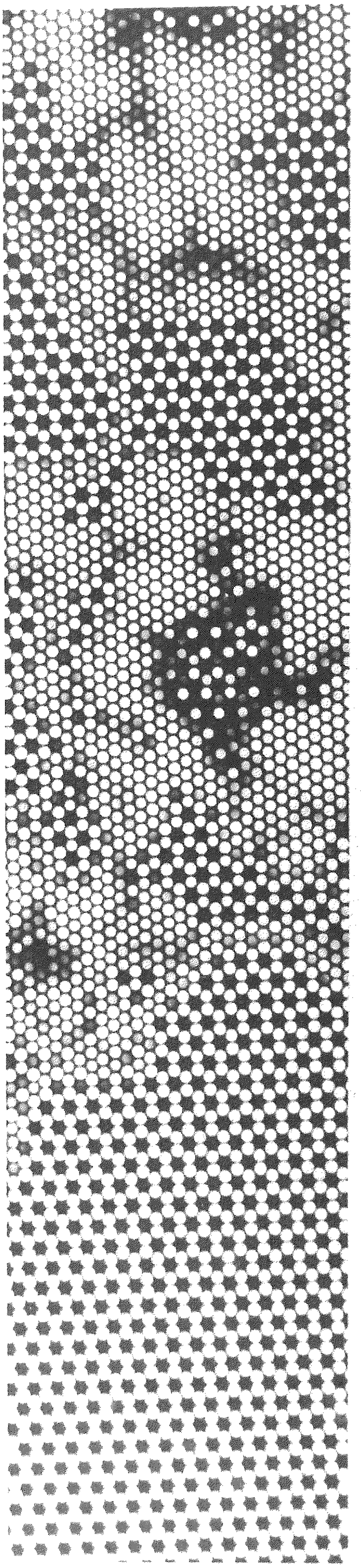


表３　DOS/Vがサポートする日本語ビデオモード番号と仕様

ALの値	名称	表示文字	文字ボックス 全角	文字ボックス 半角	画面サイズ
03H	16色文字モード	80×25	16×19	8×19	640×475
11H	2色カラー・グラフィック	80×30	16×16	8×16	640×480
12H	16色カラー・グラフィック	80×30	16×16	8×16	640×480
72H	16色カラー・グラフィック	80×25	16×19	8×19	640×480
73H	16色文字モード	80×25	16×19	8×19	640×475

※73Hの文字モードでは，「ブロック単位，属性指定つきの文字列書き込み」（INT 10H AX=131xH）の機能が使用でき，罫線なども表示できる．
※72Hの文字モードは，単に12Hのモードを25行表示に変更したものとみなすことができる．

グラフィックス用語集
VGA（Video Graphics Array）　IBM PS/2で採用されたグラフィックス用VLSI．一般にはビデオボードも含めて指す．マルチメディアプラットホームとして位置付けられる．

PROGRAMMING

GRAPHICS

特集

グラフィックス
プログラミング
の世界

ハードウェアに依存しないグラフィックス

Windowsにおける グラフィックスプログラミングの実際

中新田秀美　富士ソフトウエア株式会社

いままで，グラフィックスプログラミングといえば，機種依存するプログラムの代表的なものと思われていたが，MS-Windows や OS/2 PM の登場によって機種依存しない，マシンインディペンデントなグラフィックスプログラミングが可能になった．MS-Windows 上のグラフィックスプログラムは，GDI（Graphics Device Interface）が提供する関数を利用することで，キーボード，スクリーンの大きさ，解像度，プリンタの違いを無視することができる．つまり，グラフィックスデバイスの違いをいっさい関知する必要がないのである．このような，ハードウェアから独立した MS-Windows のグラフィックスプログラムの作成法について詳説する．

Microsoft Windows 3.0の衝撃

これまでのバージョン2.Xでは，あまり注目されていなかったMicrosoft Windowsであったが，昨年5月22日にバージョン3.0が発売されるや業界に大きな衝撃を与えた．また，この春に日本の各メーカーから相次いで日本語MS-Windows 3.0が発売され，日本でもそのボルテージはピークに達した感さえある．出荷本数は全世界ですでに400万本以上といわれ，いまも爆発的に伸びているという．MS-Windowsに対応したアプリケーション数はシステムパッケージの販売数と比較するとまだ少ないが，これから多くの優れたアプリケーションが出て（移植されて）くると思われる．

MS-DOSからMS-Windowsへ

MS-Windowsに関してこれほど世の中が騒がしくなると，これまでMS-Windowsを遠巻きにしていたMS-DOS開発者も指をくわえて見ているわけにはいかない．世の中のニーズに対応しなければならないのである．しかし，MS-DOSとMS-Windowsは基本的なコンセプトが異なるのでそれなりの準備が必要となる．ここでMS-Windowsの特徴をおさらいしてみる．

一般的ユーザーにとって，大きな特徴は次のとおりである．

・グラフィカルユーザーインターフェース

操作は，これまで記憶力，理解力，専門用語の知識に頼りがちであったが，ビジュアル化されたユーザーインターフェースによって自然な方法で簡単にコントロールできるようになった．

・アプリケーション操作の一貫性

アプリケーションの各操作はマウス操作を中心に一貫しており，安心してやさしく操作できる．また，各メーカーで開発されたアプリケーションの操作方法が統一されるため，これまでアプリケーションが変わるたびに必要とされたトレーニングの労力を大幅に軽減できる（しかし，これらは提供者側の責任によるところが大きい）．

・ハードウェア独立

アプリケーションは特定のハードウェアに依存しない．あるハードウェアのMS-Windows上で動作するアプリケーションであれば，MS-Windowsをサポートするどんなハードウェア上でも動作が保証されるのである．

開発者は，MS-Windowsによって提供されるさまざまな開発環境を効果的に利用し，効率のよい開発ができる（開発者には別に開発用キットが提供されている）．

さらに，数百種類のAPI(Applications Programming Interface)を用いることで特定のハードウェアに依存しない移植性の高いアプリケーションを構築できる．

・マルチタスク

Windowsでは複数アプリケーションが同時に動作可能である（ただし，ノンプリエンプティブなマルチタスク）．また，画面を共有(マルチウィンドウ)することで操作性のよい，ビジュアルなアプリケーションを構築できる．

・メモリ管理

Windowsは仮想メモリをサポートする．実装メモリ以上の大きさのアプリケーションを効率よく動作させるため，セグメントの移動，破棄などを行う．また，場合によってはディスクへのスワップアウト，スワップインも行う．

・ライブラリの共有

ダイナミックリンクライブラリ(DLL)によって1つのライブラリを各アプリケーションで共有できる．ライブラリの保守はコードと別に行うことができる．

・リソース

アプリケーションはシステムの提供するさまざまなリソース(アイコン，カーソル，フォントなど)を利用できる．また，リソースとコードは別々に管理できる．

アプリケーションを開発するために

アプリケーション開発者は，これまでと比べると多くのことを学ぶ必要がある．

グラフィックス用語集
VRAM(Video RAM)　画面表示する図形情報などを記憶するメモリのこと．グラフィックスを扱う場合は解像度が高くなるほど情報量も膨大になり，高速性が要求される．

Windows対応のアプリケーションを開発するために，

- イベント駆動(オブジェクト指向)のプログラミングスタイルに慣れる．
- Windowsの機能，とくにメッセージの概念を理解する．
- ハードウェアの直接的なコントロールを避ける．

などの注意が必要である．Windowsの各機能は3つのライブラリによって提供される．

- **ウィンドウマネージャインターフェース（WMI）**

ウィンドウの作成・移動・サイズ変更，メッセージ，メニューなどウィンドウ環境全般の機能を提供する．

- **システムサービスインターフェース（SSI）**

マルチタスク，メモリ，データ通信などのシステムサービスを提供する．

- **グラフィックスデバイスインターフェース（GDI）**

グラフィックスに関する出力などデバイスに依存しない出力を提供する．

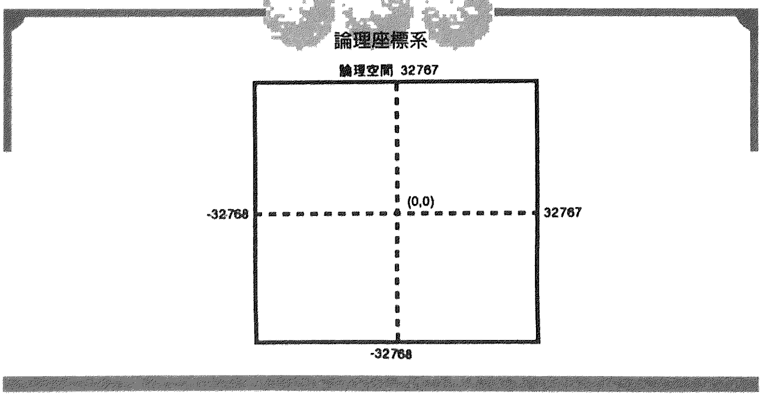
アプリケーション開発者には，これらのライブラリ機能を利用するために600種前後のAPIが提供されている．Windowsによって多くの機能がサポートされたが，ここではとくにグラフィックスを中心にそのプログラミングの実際まで見ていくことにする．

GDIとは

Graphics Device Interface (GDI) は，アプリケーションとグラフィックスデバイスとのインターフェースである．

GDIによって，各アプリケーションはグラフィックスデバイスのアクセスを共有化できる．この場合共有化はGDIが行うため，アプリケーションが共有化を意識する必要はない．また，GDIで提供される描画関数を使うことでキーボード，スクリーンの大きさ，解像度，プリンタの違いなどでプログラムの書きなおしをしないですむ．各アプリケーションは，グラフィックスデバイスの違いにいっさい関知する必要はないのである．

図1



実際に何か描画を行うには，グラフィックスに関する知識がなければならない．しかし，GDIの概念はそのほかのコンピュータグラフィックスシステムのそれと共通点が多い．ここでは，ディスプレイ出力を中心に概略に触れる．

論理座標系と物理座標系

MS-Windowsの座標系には論理座標系と物理座標系がある．

論理座標系

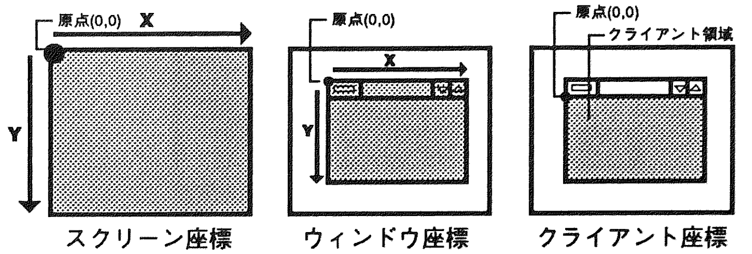
論理座標系は±32KBの範囲で利用できる（64KBのグラフ用線に相当する）．アプリケーションのプログラミングは論理座標系で行うため，デバイスの物理的な大きさを気にする必要はない（図1）．

物理座標系

物理座標系は表示領域をどこにするかによって，スクリーン座標，ウィンドウ座標，クライアント座標の3つがある(図2)．スクリーン座標はディスプレイの左上を原点(0,0)とし，ウィンドウの外側を含めたディスプレイ全体を描画対象とする．ウィンドウ座標

は，システムメニューボックス，キャプションバー，スクロールバーなどを含んだ全ウィンドウを描画対象とする．原点はウィンドウの左上である．クライアント座標は，クライアント領域内を描画対象とする．原点はクライアント領域の左上（キャプションバーのすぐ下，メニューバーがあればそのすぐ下）となる．

図2
物理座標系



マッピング

通常アプリケーションは，デバイスの大きさを意識しない論理座標系を使うことはすでに述べた．しかし，Windowsは実際に描画を行うために，論理座標を該当するデバイスのデバイス座標へ変換する．この変換方法は，マッピングモード，ウィンドウ・ビューポートの原点と範囲によって決定される．GDIでは，SetMapMode関数を使うことでマッピングモードの指定が可能である．Windowsには８つのマッピングモードが用意されている（表１）．

表１　マッピングモード

マッピングモード	論理単位	X軸	Y軸
MM_TEXT	ピクセル	左→右	上→下
MM_LOMETRIC	0.1mm	左→右	下→上
MM_HIMETRIC	0.01mm	左→右	下→上
MM_LOENGLISH	0.01in.	左→右	下→上
MM_HIENGLISH	0.001in.	左→右	下→上
MM_TWIPS	1/1440in.	左→右	下→上
MM_ISOTROPIC	任意（x=y）	選択可能	選択可能
MM_ANISOTROPIC	任意（x!=y）	選択可能	選択可能

デバイスコーディネイト

MM_TEXTはデフォルト値である．このモードは直接ピクセル単位で操作するため，デバイスによってイメージの違いがでる．また，原点（0,0）は左上であり，X座標は左から右へ，Y座標は上から下へ増加する．

物理コーディネイト

MM_LOMETRIC　(0.1mm)
MM_HIMETRIC　(0.01mm)
MM_LOENGLISH (0.01インチ)
MM_HIENGLISH (0.001インチ)
MM_TWIPS　(1/1440インチ)

アプリケーションが，物理的に意味をもつ単位で描画する場合に使用する．これらのマッピングモードは，円や正方形を正確に描画するときなどに便利である．X座標は左から右へ，Y座標は下から上へ増加する．

スクエアスケールドコーディネイト

MM_ISOTROPIC

X軸，Y軸の増減方向は任意に設定可能である．X：Yは１：１でマッピングされる．大きさは重要でないが，形に意味がある場合に使用する．写真など元のイメージが変更されては困る場合に便利である（Windowsに添付されるリバーシ，時計などはこのモードである）．

グラフィックス用語集
ビットマップ　画面上で文字なども含めグラフィックス同様にメモリ上にドットイメージとして格納すること．

ノンスクエアスケールドコーディネイト

MM_ANISOTROPIC

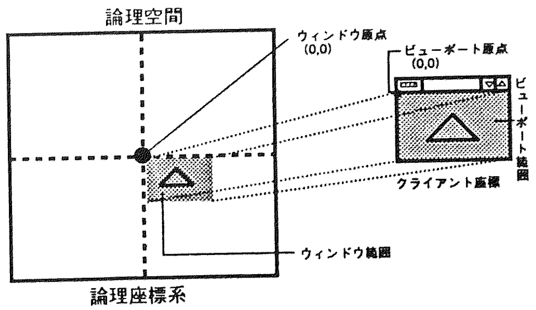
X軸，Y軸の増減方向は任意に設定可能である．X：Yのマッピングはそれぞれ独立して行われ，領域全体を有効に使用できる．イメージの形の正確性が求められる場合は不適当である．

・ウィンドウとビューポート

描画の単位はマッピングモードによってピクセル，ミリメートル，インチ，あるいはプログラムで指定した値となる．マッピングに影響を与えるもう1つの要因は，ウィンドウ(論理座標)とビューポート(物理座標)の原点と範囲である(図3)．GDIでは，論理座標系に関してSetWindowOrg関数，SetWindowExt関数などがある．また，物理座標系に関してはSetViewportOrg関数，SetViewportExt関数などが用意されている．これらの関数を使うことにより，アプリケーションは論理座標系のどこを(原点)，どれくらいの範囲で，物理座標のどこへ(原点)，どれくらいの範囲でマッピングするかを任意に設定できる．

図3

ウィンドウとビューポートの原点と範囲



デバイスコンテキスト

アプリケーションはグラフィックスを描画するとき（GDI関数を使うとき）に，必ずデバイスコンテキスト（device context:DC）を入手しなければならない．DCは，デバイスに関するさまざまな属性をもつ構造体で，GDI関数を使うときに引数として使われる．DCは，出力するデバイスごとに入手する必要がある．

ディスプレイコンテキストの型

デバイスコンテキストの中でも，とくにディスプレイに対するものをディスプレイコンテキストと呼ぶ．このディスプレイコンテキストには4種類の型がある．共通ディスプレイコンテキスト，クラスディスプレイコンテキスト，専用ディスプレイコンテキスト，ウィンドウディスプレイコンテキストである．ディスプレイコンテキストの

COLUMN 座標原点を変更する

グラフィックスで描画位置を変更する方法はいくつかある．一般には何らかのGDI描画関数で描画するX，Y座標を変更すればよい．しかし，座標系を変更することで同じ効果を出すことができる(図形の移動や反転表示など)．他のウィンドウシステムとの移植で問題となる座標原点を変更するには，SetViewPortOrg関数を使うと便利である．

座標原点を左下隅へ移動する場合，

```
SetViewPortOrg(hDC,0,Y座標の最大値)
```

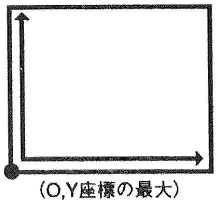
第2引数でX軸をゼロ，第3引数でY軸の値をY座標の最大値（物理座標）に指定することで座標原点を変更することが可能である（図A）．

同じように座標原点を中央へ変更するには，

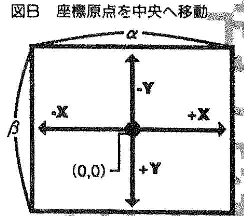
SetViewPortOrg(hDC,$\alpha/2$,$\beta/2$)

となる（図B）．

図A　座標原点を左下へ移動



図B　座標原点を中央へ移動



デフォルト値は共通ディスプレイコンテキストである.

Windowsは，システム全体で5つのディスプレイコンテキストをもつ．アプリケーションはディスプレイに描画するとき，システムのもつ5つの中から1つをキャッシュする．いま，続けてディスプレイコンテキストをもう1つ入手しようとすれば，ディスプレイコンテキストの残り4つから1つがキャッシュされる．このようにキャッシュされるディスプレイコンテキスト（共有とウィンドウ）を使う場合は，使用する数に対する注意が必要である．ディスプレイコンテキストは入手後，使い終わったら速やかに解放する必要がある.

ディスプレイコンテキストの入手

DCを入手するには，「その描画領域をどこにするか」でいくつかの方法がある.

• **BeginPaint関数**

Windowsは，ディスプレイに対して描画が必要になるとWM_PAINTメッセージを発行するが，これに対応した処理ではBeginPaint関数でDCを得なけらばならない.

この場合，描画の対象はクライアント領域となる．クライアント領域の外側に対する描画はすべてクリッピングされる.

• **GetDC関数**

WM_PAINTメッセージ以外でDCを入手する必要があるときはGetDC関数を使う．この場合もクライアント領域が描画対象となるが，BeginPaint関数とGetDC関数は内部での処理方法が異なるため，この2つの関数は明確に使い分ける必要がある.

• **WindowDC関数**

WindowDC関数で得たDCを使えば，システムメニューボックス，タイトルバーやスクロールバーを含めたクライアントウィンドウ全体に描画することができる.

• **CreateDC関数**

一般的なデバイス（プリンタなど）に関するDCを入手するにはCreateDC関数を使う．通常引数としてデバイスドライバなどを指定するが，キーワード“display”を指定すればスクリーン座標に対するディスプレイコンテキストを入手できる.

アトリビュートの設定

線画や文字を描画するとき，それらの属性はデバイスコンテキストにデフ

COLUMN　ディスプレイコンテキストの型

各アプリケーションは，1つのシステムディスプレイを共有し，それぞれのウィンドウを表示する（マルチウィンドウ)．ディスプレイコンテキストによって，アプリケーションはほかのウィンドウを気にせずに，与えられたウィンドウ内に描画することができる．ディスプレイコンテキストには，共有，クラス，専用，ウィンドウの4つの種類がある．共有，クラス，専用はクライアント領域を対象とし，ウィンドウはウィンドウ全体（タイトルバーなどを含む）を対象とする.

【共通ディスプレイコンテキスト】

共通ディスプレイコンテキストはデフォルトである．このとき，システムのもつ5つのディスプレイコンテキストから1つがキャッシュされる．アプリケーションは，DCの使用時間を必要最小限にするため，DC入手後，使い終わったらすぐに解放しなければいけない．また，DCを変更した場合その属性は残らない．各アプリケーションは，DCを取得するたびに属性を変更することになる.

【クラスディスプレイコンテキスト】

クラスディスプレイコンテキストはDCを入手後，解放する必要はない（キャッシュを使わない)．DCの内容は保存され，同じクラスに属する各ウィンドウでDCの属性を共有することができる．クラスディスプレイコンテキストを使うときはクラススタイルでCS_CLASSDCを指定する.

【専用ディスプレイコンテキスト】

専用ディスプレイコンテキストは，使用するウィンドウがはじめに一度だけ取得する（キャッシュを使わない)．そのあとDCは保存され，1つのウィンドウで独占的に何回でも使うことができる．専用ディスプレイコンテキストを使うときは，クラススタイルでCS_OWNDCを指定する.

【ウィンドウディスプレイコンテキスト】

ウィンドウディスプレイコンテキストは，タイトルバー，メニュー，スクロールバーを含め，ウィンドウ内のどこにでも描画できる．独自の境界線を描画する場合などに便利である．ディスプレイコンテキストはキャッシュを使うため，DC入手後，使い終わったらすぐに解放しなければならない．DCの内容は保存されない.

グラフィックス用語集
フォアグラウンドカラー　グラフィックス画面の点や線を表示するとき，使われる色のこと.

ォルト値として定義されている(表2). しかし, 場合によってはアプリケーションでこれらの属性を変更することもできる.

ペン

ペンの属性はCreatePen関数やCreatePenIndirect関数によって, 線の色, スタイル(実線, 破線, 点線など), 線の幅を指定する.

ブラシ

ブラシ(パタン)の属性はCreateSolidBrush関数やCreateBrushIndirect関数, CreateHatchBrush関数などによって, 色やハッチパターンを指定する.

フォント

フォントの属性はCreateFont関数, CreateFontIndirect関数などで, 文字の高さ, 幅, イタリック, 下線, ストライクアウトなどを指定する.

一般にアトリビュートを設定し利用するためには次の手順をとる.

①属性オブジェクトを生成する
　CreatePen関数, CreateSolidBrush関数など
②オブジェクトをDC内に設定する
　SelectObject関数
③描画する
　各描画関数を実行する
④変更前のオブジェクトをDC内に再設定する
　SelectObject関数
⑤生成したオブジェクトを削除する
　DeleteObject関数

表2　デフォルトデバイスコンテキストの属性と関連するGDI関数

属性	デフォルト	GDI関数
背景色	白	SetBkColor
背景モード	OPAQUE	SetBkMode
ビットマップ	デフォルトなし	CreateBitmap CreateBitmapIndirect CreateCompatibleBitmap SelectObject
ブラシ	WHITE_BRUSH	CreateBrushIndirect CreateHatchBrush CreatePatternBrush CreateSolidBrush SelectObject
ブラシ原点	(0,0)	SetBrushObject UnrealizeObject
クリッピング領域	ディスプレイ画面	ExcludeClipRect IntersectClipRect OffsetClipRgn SelectClipRgn
現在のペンの位置	(0,0)	MoveTo
描画モード	R2_COPYPEN	SetROP2
フォント	SYSTEM_FONT	CreateFont CreateFontIndirect SelectObject
内部文字スペーシング	0	SetTextCharacterExtra
マッピングモード	MM_TEXT	SetMapMode
ペン	BLACK_PEN	CreatePen CreatePenIndirect SelectObject
多角形塗りつぶしモード	ALTERNATE	SetPolyFillMode
相対－絶対フラグ	ABSOLUTE	SetRelAbs
ストレッチングモード	BLACKONWHITE	SetStretchBitMode
テキストカラー	黒	SetTextColor
ビューポート範囲	(1,1)	SetViewportExt
ビューポート原点	(0,0)	SetViewportOrg
ウィンドウ範囲	(1,1)	SetWindowExt
ウィンドウ原点	(0,0)	SetWindowOrg

プログラムの開発ツール

アプリケーション開発者は, MS-Windowsのシステムパッケージのほかにコンパイラ(MS-C 5.1以上, ま

たはQuickC 2.0以上)とソフトウェア開発キット (Software Development Kit : SDK)を揃える必要がある．また，デバイスドライバの開発者はデバイス開発キット(Device Development Kit : DDK) が必要である．

ソフトウェア開発キットには，アプリケーションの開発に必要となる多くのツールが含まれる．

また，Windowsのアプリケーション作成では，これまでのMS-DOSではあまり馴染みのない，モジュール定義ファイルやリソースファイルといった新しい入力ファイルを用意する必要がある (図4)．

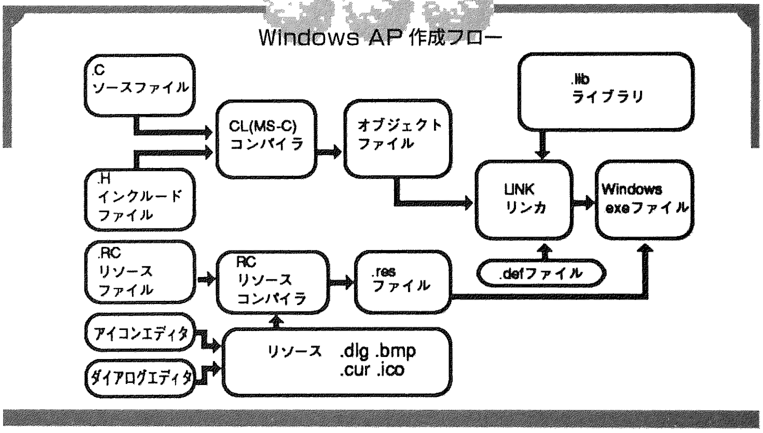
リスト1

```
NAME        SAMPLE

DESCRIPTION 'GDI sample -- Microsoft University Sample Application'

EXETYPE     WINDOWS

CODE        PRELOAD MOVEABLE DISCARDABLE
DATA        PRELOAD MOVEABLE MULTIPLE

HEAPSIZE    1024

STACKSIZE   5120

STUB        'WINSTUB.EXE'

EXPORTS     WndProc
```

リスト2

```
/* no resources */
```

図4



エディタ

Cソースを編集するためのツールとして，WindowsではWriteユーティリティ，NotePadが利用できる．また，コンパイラにはカスタマイズ可能な高機能テキストエディタとしてマイクロソフトエディタ (M.EXE) がサポートされる．

モジュール定義ファイル

モジュール定義ファイルには，リンカに与える情報を記述する．記述すべき内容は，生成するモジュール名，コード・データのセグメント属性，実行モジュールの動作タイプ，ヒープサイズ，スタックサイズ，そして，EXPORTS文によるウィンドウプロシージャ，および，コールバック関数のエクスポート宣言である (リスト1)．

リソースファイル

Windowsでは，リソースをコードと別々に管理することで保守を容易にし，移植性を上げている．ウィンドウにメニューをつけたり，ダイアログボックスを表示する場合や，また，独自のアイコンやマウスカーソルを利用する場合は，これらのリソースをリソースファイルに記述しなければならない．この情報は，リソースコンパイラ (RC.EXE) によって最終的に実行モジュールに組み込まれる．リソースとして，メニュー，アイコン，マウスカーソル，ダイアログボックス，ビットマップ，アクセラレータ，ストリングテーブル，フォントなどがある．

リソースファイルの記述方法は2つある．リソースファイルへ直接記述する方法と，専用のリソースエディタ (SDKPaint.exe, FONTEDIT.exe, Dialog.exe) でビジュアルに作成する方法である．このファイルは，たとえリソースを利用しない場合でも

グラフィックス用語集
フルカラー　写真と同等な色数を指す．1ピクセル当たり24ビットのメモリを用意し，1670万色程度を同時発色できる．

空ファイルとして必要である（リスト2）.

MAKEファイル

各コンパイル，リンク時にMAKEユーティリティを使えば，作成プロセスを自動化し，効率よく保守することができる．MAKEユーティリティはMAKEファイルを認識し，記述された情報にしたがいコンパイル，リンクを実行する．リソースコンパイラは，たとえ使用しているリソースが何もなくても実行する必要がある（リスト3）.

リスト3

```
FILE = sample

$(FILE).obj: $(FILE).c  $(FILE).m
    cl -c -AS -Gsw -Oas -Zpe /Fs $(FILE).c

$(FILE).exe: $(FILE).obj $(FILE).def
    link /NOD $(FILE), , , libw slibcew.lib, $(FILE).def
    rc $(FILE).rc  $(FILE).exe
```

テキストの出力（プログラミングの実際）

アプリケーション開発者は，Windows独自のプログラミングスタイルや，メッセージアーキテクチャに慣れなければならない．ここでは，これらのことを確認するため簡単なサンプルを用意した．このサンプルはキーボード入力，マウス入力に対応して，それぞれディスプレイへ異なる文字列を出力するものである．

アプリケーションは，最低2つの関数で構成される．1つはWinMain()関数であり，もう1つはウィンドウプロシージャ（ウィンドウ関数ともいう）である．WinMain()関数にはいくつかの初期処理とメッセージループを記述する．ウィンドウプロシージャには各メッセージに対応した処理を記述する．

サンプルソース1

行番号5

windows.hは，Windows上で使われるメッセージなどの定数，構造体の定義，システムコールのプロトタイプ宣言などが記述された重要なファイルである．100KBを超える大きなファイルであるが，アプリケーションは必ずこのファイルをインクルードしなければならない．

行番号9

文字列出力用のグローバル変数を定義する．

行番号17

WinMain()関数には4つの引数が渡される．各アプリケーションを識別するためのインスタンス，直前にアクティブにされたアプリケーションのインスタンス，コマンドラインからのパラメータ情報，初期表示に関する情報である．

行番号29〜42

最初に起動されたアプリケーションは，初期処理として必ずクラス登録を

COLUMN ソフトウェア開発キットに含まれるもの

①リソースコンパイラ（rc）
②リソースエディタ
- アイコンエディタ（SDKpaint）
- ダイアログエディタ（daialog）
- フォントエディタ（font）

③デバッグとオプティマイズツール
- コードビュー（CVW）…プロテクトモード用のマルチラインデバッガ
- シンボリックデバッガ（SYMDEB）…リアルモード用のシングルラインデバッガ
- 80386デバッガ（WDEB386）…………プロテクトモード用のシングルラインデバッガ
- スパイ（SPY）…メッセージのモニタリング
- ヒープウォーカー（Walk）…ヒープ領域の調査
- シェイカー（Shaker）…メモリ移動の調査
- プロファイラ（Profiler）…CPU時間の解析
- スワップ（Swap）…スワップの解析

④オンラインドキュメント
⑤デバッグ用カーネル
⑥サンプルソース

などが含まれる．

グラフィックス用語集
フレームバッファ　ディスプレイに表示する画像（フレーム）を蓄えるためのメモリで，VRAMのこと．

リスト4　サンプルソース1

```
/* Sample.C */
/* This is the simple Windows C Source file. */
/* Windows  GDI Programming */

#include "windows.h"

long FAR PASCAL WndProc(HWND,unsigned,WORD,LONG);

LPSTR   lpText ;                  /* common far pointer */

/*************************************************************
 *                                                   *
 *                          WinMain *
 *                                                   *
 *************************************************************/

int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine,
nCmdShow)
HANDLE hInstance;
HANDLE hPrevInstance;
LPSTR  lpszCmdLine;
int     nCmdShow;
{

WNDCLASS                  rClass;
HWND hWnd;
MSG     msg;

    if (!hPrevInstance){
        rClass.lpszClassName    = "sample";
        rClass.hInstance        = hInstance;
        rClass.lpfnWndProc      = WndProc;
        rClass.hCursor          = LoadCursor(NULL,IDC_ARROW);
        rClass.hIcon            = LoadIcon(NULL,IDI_APPLICATION);
        rClass.lpszMenuName     = NULL;
        rClass.hbrBackground    = GetStockObject(WHITE_BRUSH);
        rClass.style            = CS_HREDRAW|CS_VREDRAW;
        rClass.cbClsExtra       = 0;
        rClass.cbWndExtra       = 0;

        RegisterClass(&rClass);
    }

    hWnd = CreateWindow(  "sample",               /* class name */
                          "GDI Sample",           /* title */
                          WS_OVERLAPPEDWINDOW,    /* style */
                          CW_USEDEFAULT,          /* x potition */
                          CW_USEDEFAULT,          /* y potition */
                          CW_USEDEFAULT,          /* x size */
                          CW_USEDEFAULT,          /* y size */
                          NULL,                   /* parent */
                          NULL,                   /* menu */
                          hInstance,              /* hInstance */
                          NULL                    /* parameter */
                          );

    lpText = "Press key or either mouse button.";

    ShowWindow(hWnd,nCmdShow);
    UpdateWindow(hWnd);

    while(GetMessage(&msg,NULL,0,0)){
            TranslateMessage(&msg);
            DispatchMessage(&msg);
    }


/* Just quit this task. */
    exit(msg.wParam);

}

/*************************************************************
 *                                                   *
 *                     Window Function *
 *                                                   *
 *************************************************************/

long FAR PASCAL WndProc(hWnd,identifier,wParam,lParam)
HWND hWnd;
unsigned identifier;
WORD wParam;
long lParam;
{
HDC     hDC;
PAINTSTRUCT  ps;

    switch(identifier){

        case WM_PAINT:
                hDC=BeginPaint(hWnd,&ps);
                TextOut(hDC,20,20,lpText,_lstrlen(lpText));
                EndPaint(hWnd,&ps);
                break;

        case WM_CHAR:
                lpText = "You pressed a key.";
                InvalidateRect(hWnd,NULL,TRUE);
                break;

        case WM_LBUTTONUP:
                lpText = "You pressed the left button.";
                InvalidateRect(hWnd,NULL,TRUE);
                break;

        case WM_RBUTTONUP:
                lpText = "You pressed the right button.";
                InvalidateRect(hWnd,NULL,TRUE);
                break;

        case WM_DESTROY:
                PostQuitMessage(0);
                break;

        default:
                return(DefWindowProc(hWnd,identifier,wParam,lParam));
                break;

    }

    return(0L);
}

/* Unlike the normal strlen( ),
        this one uses a long(far) pointer */

int     _lstrlen(lpString)
LPSTR   lpString;
{
int     i;
    for(i=0;*lpString++;++i);
        return(i);
}
```

行う必要がある．クラス構造体に指定の値を設定し，ウィンドウの大まかな見てくれを設定する．中でもクラス名，ウィンドウプロシージャのアドレス，インスタンスの3つは必ず指定しなければならない．そのほかの値は，とくに指定がなければデフォルト値が設定される．クラスはRegisterClass関数で登録される．

グラフィックス用語集
ベタファイル　VRAMに格納されているメモリイメージのままのデータを圧縮などの加工なしにファイル化したもの．

行番号44〜55

生成するウィンドウの詳細を指定する．ウィンドウがどのクラスに属するか，スタイルをどうするか，どこに表示するかといったことを指定する．ここではまだウィンドウは画面に表示されない．

行番号59，60

ウィンドウを画面へ表示し，画面を更新する．ここまでがWinMain()関数で行う初期処理である．

行番号62〜65

アプリケーションはメッセージループによって，アプリケーションキューからメッセージを入手し，指定されたウィンドウプロシージャへ送る．WM_QUITメッセージが渡されるとこのループから抜けて実行を終了する．

行番号79

ウィンドウプロシージャには4つの引数が渡される．ウィンドウハンドル，メッセージIDと2つの付加情報である．付加情報は，送られるメッセージの種類によって意味あいが異なる．

行番号90

WM_PAINTメッセージ

Windowsはウィンドウの描画が必要となったとき，WM_PAINTメッセージを発行する．たとえば，ウィンドウの大きさが変更された場合（ただし，クラススタイルでCS_HREDRAW，CS_VREDRAWのフラグが設定されているとき）や，オーバラップしているウィンドウが移動し再描画が必要なときである．また，アプリケーションは，UpdateWindow関数によってWM_PAINTメッセージをいつでも強制的に発行できるが，普通は初期処理としてウィンドウが生成されたあと発行される．

行番号91

BeginPaint関数により，ディスプレイコンテキストのハンドルを入手している．以降のGDI関数ではこのDCを使うことになる．

WM_PAINTメッセージに対する処理でDCのハンドルを入手する場合は，BeginPaint関数以外（たとえばGetDC関数など）は使用してはいけない．

行番号92，93

TextOut関数で文字列をディスプレイへ出力する．引数は5つあり，それぞれ，DCのハンドル，出力する論理X座標，論理Y座標，出力する文字列，出力する文字数となる．変数lpTextはそれぞれの文字列を代入する共通変数として使われており，ここではその内容を出力している．

EndPaint関数によって入手していたDCのハンドルを解放する．

行番号96〜99

WM_CHARメッセージはキーボード入力が起こると発生する．

InvalidateRect関数は指定した矩形を更新領域に追加し，無効とする．「指定された矩形をリペイントする」といったほうがわかりやすいかもしれない．無効とマークされた領域は，次にWM_PAINTメッセージが発生したとき更新される．第2引数でNULLを指定するとクライアント領域全体を指定したことになる．

行番号101〜109

WM_LBUTTONUPメッセージ，WM_RBUTTONUPメッセージはそれぞれマウスの左ボタン，右ボタンがはなされたときに発生する．ここでも変数lpTextへそれぞれ文字列を代入している．

行番号111

WM_DESTROYメッセージは，ウィンドウが画面から削除されたあとに送られる．プログラムであと処理が必要なときなどここで行うとよい．

行番号115〜117

アプリケーションで特に処理しないメッセージは，DefWindowProc関数でシステムへ返す．デフォルト処理は必ず必要である．

行番号124〜133

_lstrlenは文字列の長さをリターン値とする関数である．

GDIの属性オブジェクトを使う

ここでは文字だけではなく線画も表示してみる．また，線のスタイルや色を変えてみる．GDIは属性として論理ペン，ブラシ（パタン），フォントなどをもつ．これらの属性は何も指定がない場合にはデフォルト値が使われる．属性オブジェクトの入手方法は2つある．

- あらかじめシステムの用意したオブジェクトを利用するときは，GetStockObject関数で使うペン，ブラシ，フォントなどを指定する．
- アプリケーション独自のオブジェク

グラフィックス用語集
ボーダーカラー　ディスプレイ画面上の表示領域の外側の色のこと．

トを生成し利用するときは，直接関数で指定する方法と構造体で指定する方法がある．

直接関数で属性を指定するときは，CreatePen関数，CreateSolidBrush関数，CreateFont関数などを使う．

構造体を使用して間接的に指定するときは，CreatePenIndirect関数，CreateBrushIndirect関数，CreateFontIndirect関数などを使う．

プログラムで属性を変更するためには次のようにする．

①属性オブジェクトを生成する，このときオブジェクトのハンドルが返される
　CreatePen関数，CreateSolidBrush関数，CreateFont関数など
②オブジェクトをDC内に設定する
　SelectObject関数
③描画する
　各描画関数を実行する
④変更前のオブジェクトをDC内に再設定する
　SelectObject関数
⑤生成したオブジェクトを削除する
　DeleteObject関数

サンプルソース2

行番号9，10

出力する文字列変数を初期化し，文字列数を18文字とする．

行番号88〜94

生成された属性オブジェクトは，各オブジェクトのハンドルを入手することで利用できる．ここでは論理ペン，ブラシ，フォントの各属性オブジェクトのためにハンドルを定義する．

行番号101

CreatePen関数には3つの引数がある．ここではペンのスタイルは点線，ペンの幅は1論理単位（この場合1ピクセル），ペンの色は赤である．より簡単な方法としてGetStockObject関数を使用してもよい．この場合はシステムで4つのペンが用意されている．

行番号104

CreateSolidBrush関数ではパターンの色を指定する．ここではグリーンを指定している．このほかにもオブジェクトとしてブラシを生成する関数には，CreateHatchBrush関数，CreatePatternBrush関数などがある．

行番号107，108

2つのストックフォントを指定している．1つは固定ピッチのシステムフォントであり，もう1つは可変ピッチのシステムフォントである．

行番号112

文字の色を赤に設定している．SetBkColor関数で背景の色を設定することもできる．

行番号116

新しいフォントのオブジェクトを設定する．このとき前に設定されていたフォントオブジェクトのハンドルが返される．

行番号119

新しいブラシのオブジェクトを設定する．やはり同じように前に設定されていたブラシオブジェクトのハンドルが返される．

リスト5　サンプルソース2

```
1 /* Sample.C */
2 /* This is the simple Windows C Source file. */
3 /* Windows  GDI Programming */
4
5 #include "windows.h"
6
7 long FAR PASCAL WndProc(HWND,unsigned,WORD,LONG);
8
9 LPSTR   lpText = "Programmer's Page." ;
10 #define STRLEN 18
11
12 /************************************************************
13  *                                                          *
14  *                                           WinMain *
15  *                                                          *
16  ************************************************************/
17
18 int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine,
19 nCmdShow)
20 HANDLE hInstance;
21 HANDLE hPrevInstance;
22 LPSTR  lpszCmdLine;
23 int     nCmdShow;
24 {
25
26 WNDCLASS                rClass;
27 HWND hWnd;
28 MSG      msg;
29
30      if (!hPrevInstance){
31
32           rClass.lpszClassName     = "sample";
33           rClass.hInstance         = hInstance;
34           rClass.lpfnWndProc       = WndProc;
35           rClass.hCursor           = LoadCursor(NULL,IDC_ARROW);
36           rClass.hIcon             = LoadIcon(NULL,IDI_APPLICATION);
37           rClass.lpszMenuName      = NULL;
38           rClass.hbrBackground     = GetStockObject(WHITE_BRUSH);
39           rClass.style             = CS_HREDRAW|CS_VREDRAW;
40           rClass.cbClsExtra        = 0;
41           rClass.cbWndExtra        = 0;
42
43           RegisterClass(&rClass);
44      }
45
46
47      hWnd = CreateWindow(   "sample",                /* class name */
48                             "GDI Sample",            /* title */
```

グラフィックス用語集
明度補正（γ補正）　個々のディスプレイや人間の目によって微妙に違うカラーマップを補正するための機能

```
49                              WS_OVERLAPPEDWINDOW,    /* style */
50                              CW_USEDEFAULT,          /* x potition */
51                              CW_USEDEFAULT,          /* y potition */
52                              CW_USEDEFAULT,          /* x size */
53                              CW_USEDEFAULT,          /* y size */
54                              NULL,                   /* parent */
55                              NULL,                   /* menu */
56                              hInstance,              /* hInstance */
57                              NULL                    /* parameter */
58                              );
59
60     ShowWindow(hWnd,nCmdShow);
61     UpdateWindow(hWnd);
62
63     while(GetMessage(&msg,NULL,0,0)){
64             TranslateMessage(&msg);
65             DispatchMessage(&msg);
66     }
67
68
69 /* Just quit this task. */
70     exit(msg.wParam);
71
72 }
73
74 /************************************************************
75  *                                                          *
76  *                      Window Function *
77  *                                                          *
78  ************************************************************/
79
80 long FAR PASCAL WndProc(hWnd,identifier,wParam,lParam)
81 HWND hWnd;
82 unsigned identifier;
83 WORD wParam;
84 long lParam;
85 {
86 HDC     hDC;
87 PAINTSTRUCT  ps;
88 HPEN    hOldPen;
89 HPEN    hDotPen;
90 HBRUSH  hOldBrush;
91 HBRUSH  hGreenBrush;
92 HFONT   hOldFont;
93 HFONT   hFontVar;
94 HFONT   hFontFixed;
95
96     switch(identifier){
97
98         case WM_PAINT:
99
100                /* Create Pen */
101                  hDotPen = CreatePen(2,1,RGB(0,0,0));
102
103                /* Create Brush */
104                  hGreenBrush = CreateSolidBrush(RGB(0,255,0));
105
106                /* Get Stock Font */
107                  hFontFixed = GetStockObject(ANSI_FIXED_FONT);
108                  hFontVar = GetStockObject(ANSI_VAR_FONT);
109
110                  hDC=BeginPaint(hWnd,&ps);
111
112                  SetTextColor(hDC,RGB(255,0,0));
113                  SelectObject(hDC, hFontVar);
114                  TextOut(hDC,20,10,lpText,STRLEN);
115
116                  hOldFont = SelectObject(hDC, hFontFixed);
117                  TextOut(hDC,20,30,lpText,STRLEN);
118
119                  hOldBrush=SelectObject(hDC,hGreenBrush);
120                  Rectangle(hDC,10,60,60,110);
121
122                  Pie(hDC,200,60,          /*      */
123                          250,110,         /*      */
124                          250,60,          /* starting point      */
125                          250,110);        /* ending point */
126
127                  hOldPen=SelectObject(hDC,hDotPen);
128                  MoveTo(hDC,10,120);
129                  LineTo(hDC,360,120);
130
131                  SelectObject(hDC,hOldBrush);
132                  SelectObject(hDC,hOldPen);
133                  SelectObject(hDC,hOldFont);
134
135                  DeleteObject(hGreenBrush);
136                  DeleteObject(hDotPen);
137                  DeleteObject(hFontVar);
138                  DeleteObject(hFontFixed);
139
140                  EndPaint(hWnd,&ps);
141                  break;
142
143         case WM_DESTROY:
144                  PostQuitMessage(0);
145                  break;
146
147         default:
148                  return(DefWindowProc(hWnd,identifier,wParam,lParam));
149                  break;
150
151         }
152
153         return(0L);
154 }
```

リスト6

```
SetMapMode(hDC,MM_ANISOTROPIC)
```

リスト7

```
SetMapMode(hDC,MM_ISOTROPIC);
SetWindowOrg(hDC,0,0);
SetWindowExt(hDC,400,200);
GetClientRect(hWnd,&rRect);
SetViewportOrg(hDC,rRect.left,rRect.top);
SetViewportExt(hDC,rRect.right,rRect.bottom);
```

リスト8

```
RECT rRect;
```

行番号120〜125

矩形と円弧を描画する．ここでは論理座標で指定する．

行番号127

新しいペンのオブジェクトを設定する．

行番号128, 129

MoveTo関数でペンの位置を移動させ，LineToで指定された位置へ線を引く．

行番号131〜133

SelectObject関数で変更前の元のオブジェクトをそれぞれDC内に再設定する．

図5

MM_ISOTROPIC

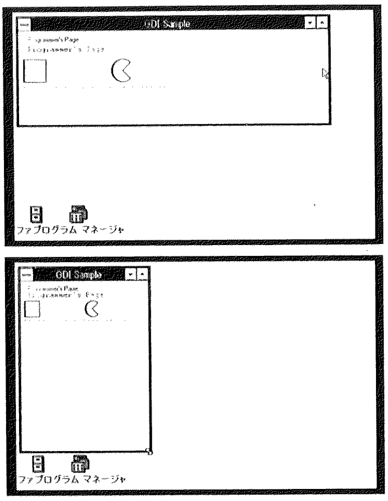
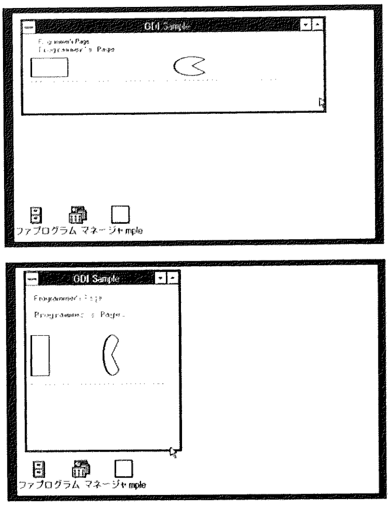


図6

MM_ANISOTROPIC



行番号135〜138

DeleteObject関数で生成されたオブジェクトをそれぞれ削除する.

さて，ここでマッピングモードの変更によって描画にどのような変化が起こるのか確認してみる. ここでは，SetMapMode関数を使いスクエアスケールドコーディネイト（MM_ISOTROPIC）とノンスクエアスケールドコーディネイト（MM_ANISOTROPIC）を確認する. プログラムに次の変更を加え，追加し，それぞれ実行結果を確認してみる.

行番号95にリスト6を追加する.

行番号110と行番号112の間にリスト7を追加する.

これでコンパイル，リンクをかける. 次にマップモードをリスト8のように変更してみる.

それぞれ実行してみると，マッピングモードによる描画の違いがよくわかる（図5，6）.

ビットマップ

ビットマップは画像をデジタル化したものであるが，Windowsでは SDKPaintユーティリティ，Paint brushユーティリティなどを使い，簡単にビットマップを作成することができる. アプリケーションでビットマップを出力するには，必ず疑似デバイスとして用意されたメモリビットマップを通じて行う. メモリビットマップの使い方はこれまでのデバイスのそれとはやや異なる. Windowsはビットマップの描画をメモリ上で展開し，最終的にデバイス（ディスプレイなど）へ転送する. メモリビットマップの使い方は次の順となる.

①デバイスコンテキストを入手する

②メモリ・デバイスコンテキストを生成する

③ビットマップオブジェクトを生成する

④オブジェクトをDC内へ設定する

⑤blt（ブリット）関数などでデバイスへビットマップを転送する

⑥変更前のオブジェクトをDC内へ再設定する

⑦生成したビットマップオブジェクトを削除する

⑧メモリ・デバイスコンテキストを削除する

となる. サンプルソースを見てみる. このサンプルではあらかじめ64×32ビットのビットマップファイル（dog.bmp）が用意されていることを前提としている. SDKPaintなどで作成するとよい. リソースとしてビットマップファイルを使用する場合は，これらの情報をリソースファイルに記述しておかなければならない（リスト9）.

WinMain()関数に変更はない.

グラフィックス用語集
レイトレーシング　光線追跡法ともいい，視点から見た物体の表面の色や輝度をドットごとに計算し表示する方法.

行番号93

WM_CREATEメッセージは，CreateWindow関数が発行されたとき一度だけ通知される．プログラムで初期処理が必要な場合にはここで対応する．

行番号94

GetWindowWord関数によって，ウィンドウのインスタンスハンドルを入手している．指定するキーワードにより，ウィンドウの親ウィンドウやタイトルへのハンドルを入手することもできる．

行番号95

ディスプレイコンテキストを入手する．ここではBeginPaint関数は使用できない．

行番号96

入手したディスプレイコンテキストとコンパチブルのメモリ・ディスプレイコンテキストを生成する．CreateCompatibleDC関数の引数は必ずあらかじめ入手された既存のディスプレイコンテキストでなけらばならない．

行番号98

LoadBitmap関数は，第2引数で指定されたビットマップリソースを実行ファイルからロードする．“dog”はリソースファイルで定義されている．

行番号99

メモリ・デバイスコンテキストにビットマップを選択する．

行番号104

BitBlt関数でメモリビットマップをディスプレイ（クライアントウィンドウ）へ転送する．第2，第3引数はクライアントウィンドウの左上の論理X，Y座標であり，第4，第5引数は幅と高さである．最後の引数はラスターオペレーション(ROP)コードであり，ビットマップ転送時の論理演算(256種類ある)を指定する．ここではそのままコピーする指定である．blt関数にはBitBlt関数のほかに1つのデバイスコンテキストを操作するPatBlt関数，ビットマップの拡大，縮小の指定ができるStretchBlt関数がある．

リスト9

```
dog     BITMAP  dog.bmp
```

リスト10　サンプルソース3

```
 1 /* sample.C */
 2 /* This is the simple Windows C Source file. */
 3 /* BitMap Display A window */
 4
 5 #include "windows.h"
 6
 7 long FAR PASCAL WndProc(HWND,unsigned,WORD,LONG);
 8
 9 /************************************************************
10  *                                                          *
11  *                         WinMain *
12  *                                                          *
13  ************************************************************/
14
15 int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine,
16 nCmdShow)
17 HANDLE hInstance;
18 HANDLE hPrevInstance;
19 LPSTR  lpszCmdLine;
20 int    nCmdShow;
21 {
22
23 WNDCLASS                    rClass;
24 HWND hWnd;
25 MSG     msg;
26
27     if (!hPrevInstance){
28
29         rClass.lpszClassName    = "sample";
30         rClass.hInstance        = hInstance;
31         rClass.lpfnWndProc      = WndProc;
32         rClass.hCursor          = LoadCursor(NULL,IDC_ARROW);
33         rClass.hIcon            = LoadIcon(NULL,IDI_APPLICATION);
34         rClass.lpszMenuName     = NULL;
35         rClass.hbrBackground    = GetStockObject(WHITE_BRUSH);
36         rClass.style            = CS_HREDRAW|CS_VREDRAW;
37         rClass.cbClsExtra       = 0;
38         rClass.cbWndExtra       = 0;
39
40         if(!RegisterClass(&rClass))
41                 return FALSE;
42     }
43
44
45     hWnd = CreateWindow(    "sample",           /* class name */
46                             "BitMap sample",
47                             WS_OVERLAPPEDWINDOW,
48                             CW_USEDEFAULT,  /* x potition */
49                             CW_USEDEFAULT,  /* y potition */
50                             CW_USEDEFAULT,  /* x size */
51                             CW_USEDEFAULT,  /* y size */
52                             NULL,           /* parent */
53                             NULL,           /* menu */
54                             hInstance,      /* hInstance */
55                             NULL            /* parameter */
56                             );
57
58     ShowWindow(hWnd,nCmdShow);
59     UpdateWindow(hWnd);
60
61     while(GetMessage(&msg,NULL,0,0)){
62             TranslateMessage(&msg);
63             DispatchMessage(&msg);
64     }
65
66
67 /* Just quit this task. */
68     exit(msg.wParam);
69
70 }
71
72 /************************************************************
73  *                                                          *
74  *                      Window Function *
75  *                                                          *
76  ************************************************************/
```

グラフィックス用語集
ローカル座標　ディスプレイ上での座標．グラフィックスをディスプレイ上に表示するときディスプレイの解像度に合わせて表示領域を計算する．

```
 77
 78 long FAR PASCAL WndProc(hWnd,identifier,wParam,lParam)
 79 HWND hWnd;
 80 unsigned identifier;
 81 WORD wParam;
 82 long lParam;
 83 {
 84 HBITMAP hBitmap ,hOldBitmap;
 85 static  HDC hMemDC;
 86 HANDLE  hMenu;
 87 HANDLE  hInst ;
 88 HDC     hDC;
 89 PAINTSTRUCT ps;
 90
 91     switch(identifier){
 92
 93         case WM_CREATE:
 94                 hInst = GetWindowWord(hWnd,GWW_HINSTANCE);
 95                 hDC = GetDC(hWnd);
 96                 hMemDC = CreateCompatibleDC(hDC);
 97                 ReleaseDC(hWnd ,hDC);
 98                 hBitmap = LoadBitmap(hInst ,"dog");
 99                 hOldBitmap = SelectObject(hMemDC ,hBitmap);
100                 break;
101
102         case WM_PAINT:
103                 hDC = BeginPaint(hWnd,&ps);
104                 BitBlt (hDC ,100 ,30 ,64 ,32 ,hMemDC ,0,0,SRCCOPY);
105                 EndPaint(hWnd ,&ps);
106                 break;
107
108         case WM_DESTROY:
109                 SelectObject(hMemDC ,hOldBitmap);
110                 DeleteDC(hMemDC);
111                 DeleteObject(hBitmap);
112
113                 PostQuitMessage(0);
114                 break;
115
116         default:
117                 return(DefWindowProc(hWnd,identifier,wParam,lParam));
118                 break;
119     }
120
121     return(0L);
122 }
```

行番号108〜114

あと処理として，メモリ・デバイスコンテキストに元のビットマップを再設定する．また，ビットマップオブジェクトとメモリ・デバイスコンテキストを削除する．

ビットマップはハードウェアにもっとも依存しやすいが，Windows 3.0ではDevice Independent BitMap (DIB) により，理論的にはデバイスに依存しないビットマップをサポートする．また，旧バージョンで作成されたビットマップを利用するときは，SDKPaintユーティリティやconvbmp.exeなどのツールでコンバートして利用する．

終りに

Windows 3.0によって，ユーザーにはビジュアルで統一されたインターフェースが提供された反面，プログラマには多くの新しい技術が要求される．プログラマは，オブジェクト指向に慣れ，メッセージ駆動システムを理解しなけらばならない．また，Windows (OS) を熟知する必要がある．さらに重要なことは，操作性を統一するためにアプリケーションプログラミングスタイルガイド (SAA) に準拠し，ハードウェアに依存しないように心がけることである．

Windows 3.0により，システム (MS-DOS) の機能は飛躍的に改善された．今後，Windowsのすばらしい機能をフルにひき出した，ユーザーフレンドリーなアプリケーションが揃うことを希望してやまない．

参考資料：

「ウィンドウズ・プログラミング」(David Durant, Geta Carlson, Paul Yao著，富士ソフトウエア訳)
「Microsoft University」(米国マイクロソフト社)
「Programming Windows」(Charles Petzold著，マイクロソフトプレス)

COLUMN　OS/2のグラフィックス(プレゼンテーションマネージャ)

MS-Windows 3.0がリリースされたとき，世間では下位バージョンからの移行方法が話題となったが，ここ日本ではMS OS/2からWindowsの移行が大きな話題となった．これは日本と米国の市場の違いとして興味深いできごとである．現在，開発者にとってシステムがMS-WindowsかOS/2かということは大きな問題となる．

OS/2のグラフィックス(PM)は，グラフィックプログラミングインターフェース (GPI) という形でサポートされている．GPIはWindowsのGDIと似ているが，プログラミングレベルの細部では異なる．とくにウィンドウイング，座標形，GPI関数などである．

PMでは，ウィンドウを構成するシステムメニューボックス，タイトルバーなどとクライアント領域は同レベルに位置づけられる (Main関数で設定するクラス登録，ウィンドウの生成などの初期処理も異なる)．また，PMの座標原点は左下隅であり，Windowsは右上隅である(プログラム側で対応はできるが)．さらに，サポートされる関数名はWindowsのそれとは異なる．

しかしながら，今後MS-WindowsにOS/2のグラフィック機能が追加されていくようであるので，システムの違いを意識する必要は少なくなるものと思われる．

グラフィックス用語集
ワイプ　次の画像を表示するとき，現在の画像を押し出すようにして表示する方法

特集 グラフィックスプログラミングの世界

市販フレームバッファ グラフィックスライブラリ 製品紹介

フレームバッファ

HyPER-FRAME+

HyPER-FRAME+は，(株) デジタルアーツが販売している16,777,216色表示が可能なPC-9801用フルカラーフレームバッファである．

HyPER-FRAME+の特徴的な機能として，

- パソコン本体の映像出力（テキスト，グラフィックス）とフレームバッファの映像を重ね合わせるスーパーインポーズ機能．
- スクウェアウィンドウとプログラマブルウィンドウの2種類のウィンドウを備え，フレームバッファ出力とパソコン出力を選択して表示できる．
- 32×32ドットのカーソルパターンを32個まで登録できる．
- 縦横独立して1，2，4，8倍の拡大ができるズーム機能．
- RGB各8ビットのビット数（量子化ビット数）を4段階に設定できる．
- V-RAMの内容を変えることなく，RGB独立に表示色を調整できる．
- データの入出力を，XY座標指定のI/O方式とI・O方式のバンク切り替え方式とを切り替えて使用できる．

などがある．

HyPER-FRAME+には，ユーザーがHyPER-FRAME+の機能を利用したプログラムを作成するのに利用できるN88-日本語BASIC(86)，N88-日本語BASIC (MS-DOS版) 用マシン語ルーチン，およびMS-C，Turbo C，ZORTECH-C用C言語ライブラリ（スモールモデル用／ラージモデル用）が付属している（表1）．

製品名：HyPER-FRAME+
対応機種：PC-9801シリーズ／PC-286・PC-386シリーズ　※アナログRGB端子のある機種に限る．PC-98XL/XL²/RLはノーマルモードで動作．H98は別売ケーブルHF2が必要．
価格：98,000円
問い合わせ先：（株）デジタルアーツ
TEL：03-3486-9697 (インフォメーションサービス)

写像

写像は，高機能レイトレーシングソフトウェア「C-TRACE」で知られる(株)キャストが販売している低価格なPC-9801用フルカラーフレームバッファである．

写像の機能としては，

- PC-9801のRGB画面と写像のフルカラーの画面をスーパーインポーズすることが可能．
- I/Oポートを介したインターフェースのため，CPUのメモリ空間を圧迫しない．
- 写像へのアクセスはサイクルスチールのため，タイミング管理を考えずにメモリに読み書き可能．
- PC-9801シリーズと同じX-Y座標によるメモリ構成のため，アドレス計算が不要．
- Xレジスタ，Yレジスタのオートインクリメントにより高速データ転送を実現．
- 1，2，4，8倍のハードウェアズーム，縦横独立な1ピクセル単位でのスクロール（パン）機能．

などがある．

また，プログラミングを行うユーザーのためにC言語で書かれた写像アク

表1　HyPER-FRAME+付属C言語ライブラリ主要関数一覧

関数	機能
HFOn()	HyPER-FRAME+使用の宣言
HFOff()	〃
HFSORE()	ソフトウェアリセットの設定
HFBnlo()	アクセス方式の設定
HFRgIni()	レジスタのイニシャライズ
HFRgSw()	レジスタ機能のオン／オフ
HFDisp()	画面表示の切り替え
HFAnDg()	アナログ16色表示の選択
HFClr()	画面のクリア
HFPSet()	点を打つ
HFPGet()	点の色を読み取る
HFLine()	直線を引く
HFXLine()	画面上のデータとXORをとって直線を引く
HFBox()	四角形を描く
HFBoxFil()	塗りつぶした四角形を描く
HFEllipse()	楕円(円)を描く
HFPaint()	閉じた線の内部を塗りつぶす
HFWmSe()	ライトマスク機能のセット
HFWmSw()	ライトマスクスイッチのセット
HFCsrSw()	カーソル表示のオン／オフ
HFCsrNo()	表示するカーソル番号の選択
HFCsrPoi()	カーソル表示位置の変更
HFCsrCol()	表示するカーソル色の変更
HFZoom()	ズーム機能の設定
HFScroll()	画面のスクロール
HFQuant1()	ビットスライスグラフィックスの設定（CH1/CH2）
HFQuant2()	〃
HFLoad()	画像ファイルのロード
HFSave()	画像ファイルのセーブ

グラフィックス用語集
ワイヤーフレーム　3次元グラフィックスなどで物体を点と点を結ぶ線（ワイヤー）の集合として表現する方法

セスルーチンが付属している.

製品名：写像
対応機種：PC-9801シリーズ
価格：69,800円
問い合わせ先：(株)キャスト
TEL：03-3705-1065

SuperFrame²/SuperFrame²Σ

(株) サピエンスが販売しているSuperFrame²/SuperFrame²Σは，SuperFrameから続くサピエンスアーキテクチャを継承するフルカラーフレームバッファである.

サピエンスアーキテクチャと呼ばれるSuperFrame²/SuperFrame²Σの特徴には，

- PC-9801の画面とフルカラーの画面をスーパーインポーズすることで，マウスカーソルを表示するなどPC-9801の画面を利用できる.
- ビデオ用デュアルポートRAMの採用により，CPUはいつでもフレームメモリにアクセスできる.
- CPUのフレームメモリへのアクセス単位がピクセル単位であるため，ビットシフト演算，ビットマスク演算が不要.
- 2次元座標系CRTコントローラを採用したため，ピクセルのアドレス計算が不要.
- 完全なI/O型フレームメモリなので，PC-9801のメインメモリ空間を圧迫しない.
- 4画素同時書き込みが可能.
- 座標レジスタの自動インクリメント
- 画素アクセス後のリカバリタイムが0nsecなので，CPUがどんなに高速になっても対応可能.
- 画像データのファイル形式は，CGプリントサービスのファイル形式と同一.

などがある.

また，SuperFrame²/SuperFrame²Σを利用したプログラム作成のために，MS-C Ver.3.0以上，Lattice C Ver.J3.11, Optimizing C86 Ver.2.30J用のC言語グラフィックスライブラリ（スモールコードモデル用／ビッグコードモデル用）が付属している(表2). このライブラリを使用して作成されたプログラムは，そのプログラムがSuperFrame²/SuperFrame²Σ用であるかぎり，自由に配布／販売することができる.

製品名：SuperFrame²/SuperFrame²Σ
対応機種：PC-9801VM/VX/UV/UX/ES/EX/RX/RA/RS/DA/DS/DXシリーズ，PC-98XL/XL²/RL/H98(ノーマルモード)シリーズ
価格：SuperFrame²（型番SP98451）158,000円／2枚目用SuperFrame²（型番SP98452）158,000円／SuperFrame²Σ(型番SP98454) 178,000円／SuperFrame²Σ（H98S対応：型番SP98454X)178,000円／SuperFrame²Σ(H98対応：型番SP98454X-H) 178,000円
問い合わせ先：(株)サピエンス
TEL：03-5379-8771

グラフィックスライブラリ

Bit Map Text & Graphics Adaptor Ver.1.0

Bit Map Text & Graphics Adaptor Ver.1.0は，MS-C Ver.5.1およびVer.6.0に対応したPC-9801用グラフィックス処理ライブラリである. Bit Map Text & Graphics Adaptor Ver.1.0のおもな特徴は，

- 通常の16色に加えてハーフトーンを重ね合わせた136色グラフィックスの同時発色が可能.
- グラフィックスについては，通常必要とされる機能はすべて備えている. キューブ・パイプ・多角形の描画，フェードイン，フェードアウト，ブラインド，裏表画面入れ換え，画面のハードコピーなどが可能で，さらにグラフ機能として折れ線グラフ，立体棒グラフ，円グラフなどを使用できる.
- 縦横各1/2～8倍の縮小・拡大，1/4サイズ，極小サイズの描画が可能で，さらに拡大時には粗さを補正するスムージング機能が働く.
- 上下逆，左右逆，横倒し，縦書きなど，自由に文字方向を変更できる. さらに文字については，11.25度ごとの傾斜，袋文字，影文字などの装飾が行える.
- PC-9801本体のスピーカーを利用した簡易自動演奏機能.

表2　SuperFrame²/SuperFrame²Σ付属C言語グラフィックスライブラリ関数一覧

関数	機能
initsf()	スーパーフレームの初期設定
pset()	指定画素に色を設定する
spset()	〃
maxpset()	〃
minpset()	〃
redset()	指定画素の赤色をセットする
greenset()	指定画素の緑色をセットする
blueset()	指定画素の青色をセットする
redget()	指定画素の赤色の値を得る
greenget()	指定画素の緑色の値を得る
blueget()	指定画素の青色の値を得る
line0()	直線を描く
line1()	〃
circle0()	円を描く
circle1()	〃
arc0()	弧を描く
arc1()	〃

グラフィックス用語集
リサージュ図形　互いに直行するふたつの単振動の合成よって作られる図形.

のようになっている．

また姉妹品として，MS-DOS上で簡単にマルチタスク処理プログラムの開発が行えるMULTI TASK ADAPTOR Ver.2.0（価格：49,800円）が発売されている．これらのライブラリを使用して作成されたプログラムは，ライセンスフリーで販売・配布することが可能である．

(株)リンクスアイではデモソフトの無料配布を行っているので，興味のある方は問い合わせてみるとよいだろう．

製品名：Bit Map Text & Graphics Adaptor Ver.1.0
対応機種：PC-9801シリーズ
価格：58,000円
問い合わせ先：(株)リンクスアイ
TEL：03-3944-2295

GENIFA

本誌7月号「Programmer's Program」でも紹介した，グラフィックス・GUI開発ツール&ライブラリである．

GENIFAのおもな特徴は，

- 異なる機種（PC-9800シリーズおよび互換機，AX，IBM PC/AT（EGA，VGA）および互換機）の間で高い互換性を実現している．
- グラフィックス描画ルーチンは，アセンブラによるハードウェアの直接制御に加えBresenhamアルゴリズムの採用によって，高速描画を実現している．
- プルダウンメニューエディタ，ダイアログボックスエディタ，マウスカーソルエディタ，フォントエディタ，アイコンエディタなどのユーティリティを装備．
- グラフィックスソフト開発に必要な約100種類のグラフィックス機能を搭載．

となっている．

なお，GENIFAはMS-C Ver.5.1以上に対応している．

製品名：GENIFA
対応機種：PC-9801，AX，IBM PC/AT（EGA，VGA）
価格：131,000円（シングルユース）
問い合わせ先：(株)アイ・エル・シー
TEL：082-261-9933

UCG(Ultra C Graph)Ver.2.0

UCGは，MS-C Ver.5.1，QuickC Ver.2.0，Turbo C Ver.2.0，Lattice C Ver.4.1などに対応したPC-9801，PC-286用グラフィックスライブラリである．

UCGのおもな特徴は，

- コンパイラがサポートしているすべてのメモリモデルに対応している．
- アセンブラで記述されたサブルーチンにより，主要な描画コマンドにおいて，REPLACE，COMPLEMENT，CLEAR，SETの描画モードをサポート．
- REPLACEモード時は，ROMを使用してEGCのスピードを100%引き出す．
- ANSI準拠のC言語に対応しており，UCGが含むcgraph.h，cmouse.hなどのヘッダファイルではプロトタイプ宣言がされている．
- テキスト画面をグラフィックス画面にコピーしたり，ファイルをグラフィックス画面にダンプしたり，Z's STAFFのグラフィックスファイルをロードするなどのユーティリティが付属している．

などである．

また，UCGを使用したソフトウェアを販売したり多人数で使用するときのために，Ultra C Graph Pro（コーポレイトライセンス版）が用意されている．

製品名：UCG(Ultra C Graph) Ver.2.0
対応機種：PC-9801
価格：32,000円（パーソナルライセンス版）／180,000円（コーポレイトライセンス版）
問い合わせ先：(株)アークブレイン
TEL：03-3375-8968

Graphic & Mouse(/EX)

Graphic & Mouse(/EX)は，N88 BASIC相当のグラフィックス機能とマウス機能を提供するPC-9800シリーズ用C言語ライブラリである．対応しているコンパイラは，MS-C Ver.5.1/6.0，QuickC Ver.2.0，Turbo C Ver.2.0/C++で，各コンパイラごとに対応した製品が用意されている．

Graphic & Mouse（/EX）のおもな特徴は，

- 拡張グラフィックス関数は，REPLACE，CLEAR，OR，XORのラスタオペレーションが可能（ノーマルモードのみ）．
- PC-9801XL，XL²，RLのノーマルモード，ハイレゾリューションモードの両モードに対応している．ハイレゾリューションモード時に互換モードを指定することで，プログラム変更なしにノーマルモードと同様な描画が得られる．
- 各関数は，ライブラリファイルおよびソースコードの形で供給される．

などである．

Graphic & Mouse（/EX）の組み込みロイヤルティは無料である．ただし，第三者がGraphic & Mouse（/EX）に含まれる関数やソースを直接利用可能な形での再販については，別途契約が必要となる．

製品名：Graphic & Mouse（/EX）
対応機種：PC-9800シリーズ（/EXはハイレゾリューションマシンにも対応）
価格：Graphic & Mouse　29,800円／Graphic & Mouse/EX　59,800円
問い合わせ先：(株)ティップス
TEL：03-5828-0511

グラフィックス用語集
フラクタル図形　全体の中の部分が全体と同じ形である自己相似性を持った図形．

パソコン博物館 IX

インテリジェントペリフェラルコントローラ

BASIC MASTER LEVEL3

日立　ベーシックマスターレベル3

井戸禎光

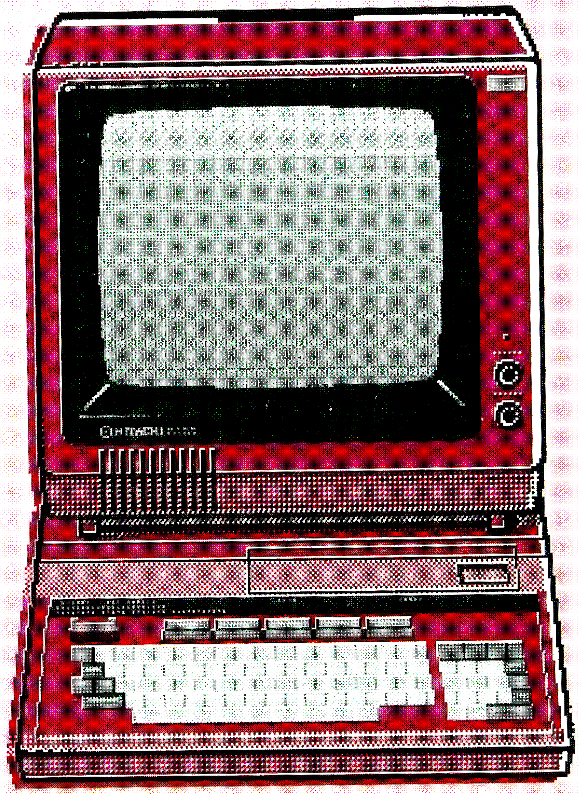


Illustration
TOSHIHITO ITO

1980年，日立はMB-6890ベーシックマスターレベル3（298,000円）を発表した．名前はパソコンらしからぬものであったが，非常に人気が高いマシンであった．CPUには，当時究極の8ビットCPUといわれていた6809を使い，その高速性能が売りものであった．もともとの本機のコンセプトは名前の通り，ずばり“BASICソフトをマスターすること”であり，強力なLEVEL-3BASICを備えていた．CRTは，640×200という標準仕様になっていたし，パソコン初のひらがなが，あるいは7色のカラーが表示可能であった．外部記憶装置は，オーディオカセットテープ，8インチ両面倍密度，および5.25インチの片面単密度や両面倍密度のフロッピーディスクが付けられるようになっていた．

日立は，当時H68シリーズという，CPUに6800を使ったトレーニングモジュールを出していた．この部分は99,500円であったが，キーボードが28,000円，電源が5V-10Aで19,000円，+5V-+/-12V1Aで35,000円，カラーテレビインターフェースモジュールにいたっては89,500円という時代であった．これで一式組み立てていては相当高いものになってしまう．

1978年になると，CPUに同じく6800（日立名46800）を採用したMB-6880初代ベーシックマスターを出し，価格は188,000円とした．続いてRAMを4KBから32KBに拡張し，BASICも整数型から浮動小数点型に拡張したMB-6881ベーシックマスターレベル2となり，価格は，148,000円となった．この機械は，名前の通りに編集コマンドとエディタを内蔵していたし，モニタの機能は非常に強力，三角関数・自然対数・平方根などの多種類の関数も内蔵していて，ディスプレイ（498,000円），カセットレコーダ（14,800円）などを接続するのみで，ホビーユースとしては十分楽しめるということで，人気が出たものであった．

1980年当時の競合品は，外国勢としては，S-100バスのアルテア8800，コモドール3000シリーズ，アップルII，TRS-80，国産勢としてはNECのPC-8001，シャープのMZ-80C，それに価格帯は少し上であったが，SORDのM-200があった．この時の競合において，多くの機器はザイログ社のZ-80をCPUに使っていた．

それに対し日立は，従来からモトローラの6800を使っていたが，上記の競合状態においても，同じくモトローラの6809を採用したのだった．6809は8ビット系における最高のCPUであった．Z-80がいろいろな欠点をもちながらも，実際にパソコンを作る段になると別の長所を表すため，根強い人気をもっているのに対し，6809はアーキテクチャのシンプルさと大型コンピュータと同じソフトウェア構成を可能にするアーキテクチャと高速性を売りものに登場していた．

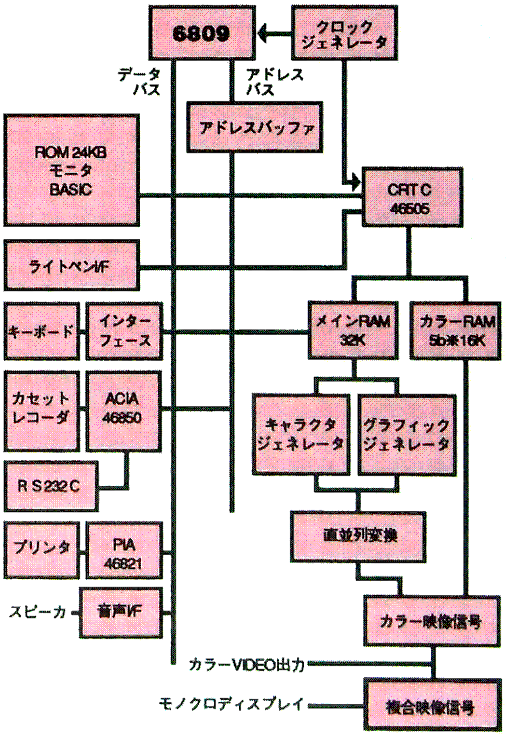
また，インターフェースの面でも，レベル3となるとライトペン，プリンタ，カセットテープ，RS-232C用のものが標準で付いていた．H68のインターフェースでは，TV用のものだけでも約70,000円といっていたが，このような時代が過ぎ去っていったのである．

ハードウェアの回路上も今とは異なっていて，興味深い点が多い．全回路の中でもっとも高速のクロックを必要とするのは，当時は映像信号の中にあるドット信号であった．すなわち，NTSC方式で水平の走査線1本は約1÷(60×256)=63μ sec，この中に640ドット入れると（横40文字で文字当たり8ドット，文字間隔を2ドットとすると400ドット程度必要），ドット当たり100nsecとなるが，輝点が右から左にかえる時間を考えて，ベーシックマスターレベル2では，12MHzを発振させて1/2分周し，83nsecの信号を作り，1ドットとしている．

キャラクタジェネレータROM（2316，16Kビット）に入れてある文字ドットの8ビット信号を上記のクロック信号を用いてシリアル信号になおしCRTに送る．このとき，CRT画面の右端にきたら，次の字は左端にするとかいろいろな制御が必要になる．レベル2ではCRTの制御はTTLで行い，カセットテープはソフトウェアタイマで信号波形を作っていた．レベル3になるとハードウェアで行わせるようになって，“価格低減と高速化”を両立させることができるようになった．とくに，CRTコントローラは6800用のCRTCとして開発されたが，以後広い範囲で長年使われているHD46505という有名なICである．

このような点は，技術の日立の面目躍如というところであるが，B16，B32を経て，最近AX仕様のパソコンを出すことになった．どんな新技術を開発して参入するか，今から楽しみである．

ベーシックマスターレベル3の構成



祐安重夫

プログラマの道具学

第9回　シンセサイザ

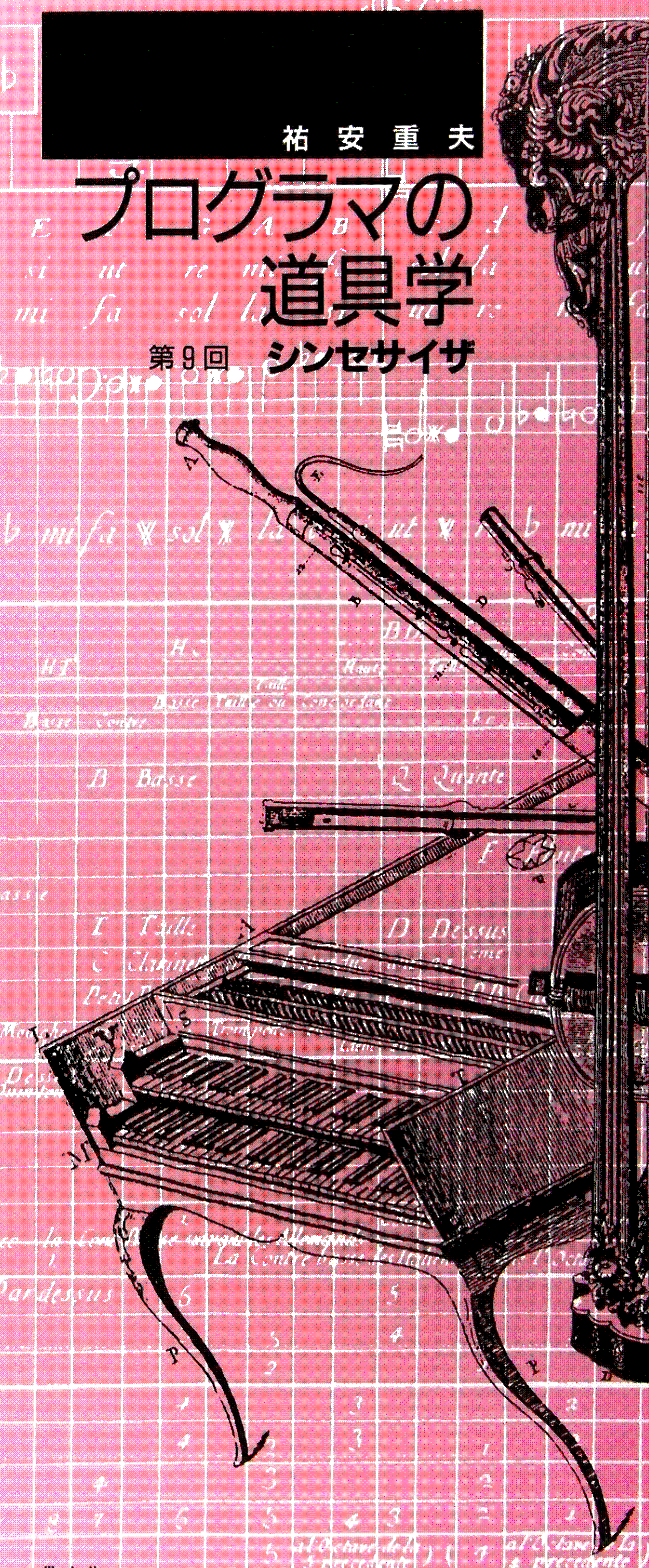


Illustration
Mikio Sasaki

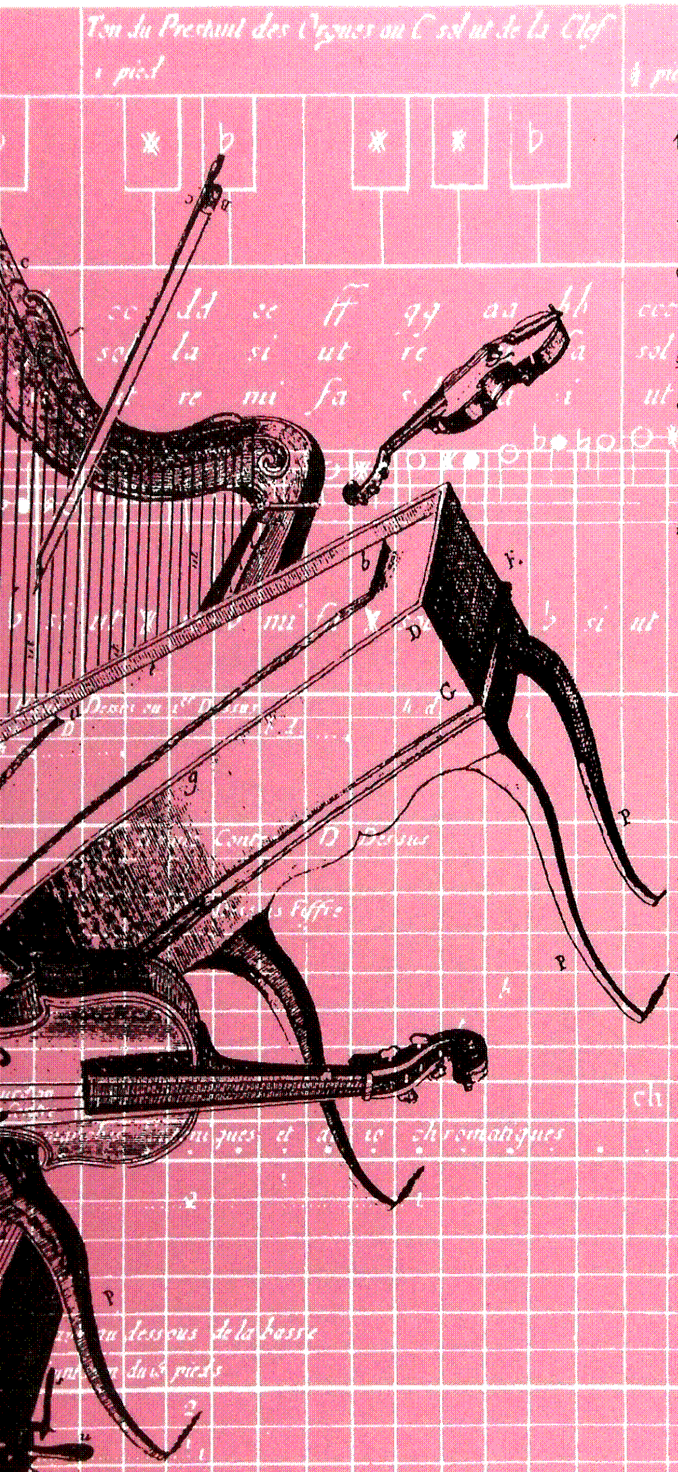
もう15年以上も以前のことになるが，ソフトウェアの仕事をはじめる前は音楽を仕事にしていた．音楽学校で当時はまだめずらしかったシンセサイザの講座の講師をしていたこともあって，今でも机の周囲にあるコンピュータのうち4台が，音楽機能をもっている．もっともそういう経歴がなくても，コンピュータにシンセサイザをつないでいる例は，かなりおおいようだ．

4台のうち1台はまったく独立したシステムになっているが，他の3台はそれぞれ別のコンピュータとつながって，相互に情報のやりとりができるようになっている．本格的なLAN (Local Area Network) になっているのは，Unixマシンを中心とした一部分だけだが，それ以外のコンピュータもRS-232Cによって相互につながっているので，ソフトウェアを用意してやれば，1台のコンピュータから他のコンピュータをコントロールすることが可能である．

その昔，はじめてシンセサイザを使用した頃は，電圧制御型とよばれる方式が主流をしめていた．これは音高や音価といったパラメータを電圧の変化によってコントロールするもので，コンピュータからはDAコンバータをつかえば比較的簡単に制御できた．

現在ではMIDI (Musical Instrument Digital Interface) によって，ほとんどの電子楽器が標準的な方法でコンピュータと接続可能になっているが，昔とくらべていささかおもしろみがなくなってきているような気がするのは，ないものねだりだろうか．

私のところでも最初のうちは，MIDIによってシンセサイザなどの音源をつないだコンピュータで，それぞれにソフトウェアをつくり，音楽データを用意したり作曲プログラムで自動演奏をおこなっていたのだが，簡単に音源が接続できたり，音源を内蔵しているような



システムは，かならずしもつかい勝手がいいとはいえなかった．また市販の音楽ソフトウェアは，あまりおもしろいものではなかったのもたしかである．ただMacintoshのソフトウェアであるMaxはかなりおもしろそうなのだが，まだつかう機会をえていない．

今では音源をもったコンピュータには，最低限のシンセサイザのコントロールと，別のコンピュータからの情報をうけとるためのソフトウェアだけを用意し，実際の音楽データの処理はUnixでおこなうような環境を準備した．Unixならばソフトウェアをつくるのが楽だし，大量の音楽データを保存しておくためのディスク領域も，比較的確保しやすい．また操作性にもすぐれている．Unixの欠点はリアルタイムの制御ができない点だが，時間管理を音源側のコンピュータにやらせることで，なんとか解決できる．

こうして今では，Unixワークステーションのマルチ・ウィンドウ環境で仕事をしながら，さらにBGMのコントロールまでおこなうということもできるようになっている．ただしシンセサイザの音は，あまり長時間きいているとあきてくるので，カセットやCDで普通の音楽テープをながしていることもある．

そういえば最近ではコンピュータにCD-ROMがつながるようになってきており，それがCDプレーヤとしてもつかえるようである．実際にCDプレーヤのパネルをX-Windowの画面にだして，それをマウスでコントロールするソフトウェアが，現在つかっているワークステーションに付属している．もっともCD-ROMのドライブをもっていないので，まだつかってみたことはない．

すでにMacintoshやNeXTのようなコンピュータではかなりのレベルまで実現されてきているが，Unixワークステーションにも本格的なAV機能をもたせようとするうごきが，いくつかでてきている．もともとはマルチ・メディア・データベースやハイパー・メディアとよばれるものをターゲットとしているのだろうが，それを音楽につかっていけないということはない．その意味では，これからがたのしみである．

プログラム遊工房

谷口典生

九乃巻
「射撃」ゲーム

今回はアクションゲームに挑戦してみました．シューティングゲームの原点のような単純なゲームではありますが，動きのあるゲームは見ているだけで楽しいものです．
ワンポイントテクニックでは，アクションゲームのプログラムの基本的な構造について解説してみました．これを参考にプログラムを改造してオリジナルのゲームを作ってみてはどうでしょうか．

リスト1

```
/*
 * shoot.c              射撃ゲーム
 */

#include <time.h>
#include <math.h>
#include <stdlib.h>
#ifdef __TURBOC__
/* Turbo C 用グラフィック初期化/復帰ルーチン */
#include <graphics.h>
void graphinit()                        /* グラフィックモード設定 */
{
        int graphdriver, graphmode;

        graphdriver = DETECT;           /* 自動検出 */
        graphmode = PC98C8;             /* 8色モード */
        initgraph(&graphdriver, &graphmode, "");
}
void graphdeinit()                      /* グラフィックモード復帰 */
{
        restorecrtmode();               /* 画面モードを元に戻す */
        printf("¥x1b*");                /* 画面クリア */
}
#else
/* MS-C, Quick C 用グラフィック初期化/復帰ルーチン */
#include <graph.h>
#define BLACK                   0
#define BLUE                    1
#define GREEN                   2
#define CYAN                    3
#define RED                     4
#define MAGENTA                 5
#define YELLOW                  6
#define WHITE                   7
#define setcolor                _setcolor
#define moveto(x,y)             _moveto(x,y)
#define lineto(x,y)             _lineto(x,y)
#define rectangle(l,t,r,b)      _rectangle(_GBORDER,l,t,r,b)
void graphinit()                        /* グラフィックモード設定 */
{
        _setvideomode(_98RESSCOLOR);    /* グラフィック8色+テキストモード */
        printf("¥x1b[>5h");             /* カーソル消去 */
}
void graphdeinit()                      /* グラフィックモード復帰 */
{
        _setvideomode(_DEFAULTMODE);    /* デフォルト(テキスト80行)モード */
        printf("¥x1b[>5l¥x1b[>1l");     /* カーソル表示、最下行復帰 */
}
#endif

#define LAND_Y          399             /* 地面の位置 */
#define CANNON_SIZE     28              /* 砲身の長さ */
#define MAX_SHOTS       10              /* 砲弾の最大数 */
#define MAX_BOMBERS     10              /* 爆撃機の最大数 */

struct shotstruct {                     /* 砲弾の状態を定義する構造体 */
        int stat, x, y;                 /* 弾の状態、XY座標 */
        int speed_x, speed_y;           /* 弾の速度のXY成分 */
} shot[MAX_SHOTS];

struct bomberstruct {                   /* 爆撃機の状態を定義する構造体 */
        int stat, x, y, speed;          /* 爆撃機の状態、XY座標、速度(X方向) */
        int bstat, bx, by, bspeed;      /* 爆弾の状態、XY座標、速度(Y方向) */
} bomber[MAX_BOMBERS];

int cannon_pos, cannon_speed;           /* 砲台の位置、砲台の速度 */
double cannon_dir, cannon_dd;           /* 砲身の向き、速度 */
int cannon_dx, cannon_dy;               /* 砲身の向きのXY成分 */
int cannon_destroyed=0;                 /* 砲台に敵弾があたったか */
int score=0;                            /* 得点 */

void putbomber(int n, int color)        /* 爆撃機の表示 */
{
        setcolor(color);
        moveto(bomber[n].x-15, bomber[n].y);
        lineto(bomber[n].x+15, bomber[n].y);
        lineto(bomber[n].x-15, bomber[n].y-10);
        lineto(bomber[n].x-15, bomber[n].y);
}

void putcannon(int color)               /* 砲台の表示 */
{
        setcolor(color);
        rectangle(cannon_pos-15, LAND_Y-10, cannon_pos+15, LAND_Y-1);
        moveto(cannon_pos, LAND_Y-10);
        lineto(cannon_pos+cannon_dx, LAND_Y-10-cannon_dy);
}

void putshot(int n, int color)          /* 砲弾の表示 */
{
        setcolor(color);
        rectangle(shot[n].x-1, shot[n].y-1, shot[n].x+1, shot[n].y+1);
}
```

プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C, QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

今回のプログラムではグラフィックス関数を使っているので，MS-CやQuickCでコンパイルするときにはグラフィックスライブラリを指定します．MS-Cなら「CL shoot.c /link graphics.lib」などとします．Turbo Cでは実行時に.BGIファイルが必要となるので注意してください．

プログラムの実行は，

SHOOT {wait_time}

と，適当な待ち時間を指定します．V30 8MHzでは0（ウエイトなし），80286 10MHzで4，80386 20MHzで10くらいの値が適当だと思います．

```
 94:
 95: void putbomb(int n, int color)          /* 爆弾の表示 */
 96: {
 97:         setcolor(color);
 98:         rectangle(bomber[n].bx-1, bomber[n].by-1, bomber[n].bx+1, bomber[n].by+1);
 99: }
100:
101: void shoot()                            /* 弾を撃つ */
102: {
103:         int i;
104:
105:         for(i=0; i<MAX_SHOTS; i++)              /* 弾データの空きを捜す */
106:                 if(shot[i].stat==0)
107:                         break;
108:         if(i==MAX_SHOTS)                        /* 弾データが空いていない */
109:                 return;
110:         shot[i].stat = 1;
111:         shot[i].x = cannon_pos;
112:         shot[i].y = LAND_Y-10;
113:         shot[i].speed_x = cannon_dx;
114:         shot[i].speed_y = -cannon_dy;
115: }
116:
117: void movecannon()                       /* 砲台を動かす */
118: {
119:         int i,key;
120:
121:         if(kbhit()) {
122:                 key = getch();
123:                 switch(toupper(key)) {
124:                 case 'H':                       /* 左移動 */
125:                 case '4':
126:                         cannon_speed = -2;
127:                         break;
128:                 case 'L':                       /* 右移動 */
129:                 case '6':
130:                         cannon_speed = 2;
131:                         break;
132:                 case ' ':                       /* 弾を撃つ */
133:                         shoot();
134:                         break;
135:                 case 0x1b:                      /* 自爆 */
136:                         cannon_destroyed = 1;
137:                         break;
138:                 default:                        /* 止まる */
139:                         cannon_speed = 0;
140:                 }
141:         }
142:         putcannon(BLACK);                       /* 砲台を消す */
143:         cannon_dir += cannon_dd;
144:         if(cannon_dir<0.5 && cannon_dd<0 || cannon_dir>2.6 && cannon_dd>0)
145:                 cannon_dd = -cannon_dd;
146:         cannon_dx = cos(cannon_dir)*CANNON_SIZE;
147:         cannon_dy = sin(cannon_dir)*CANNON_SIZE;
148:         if(cannon_pos+cannon_speed>=100 && cannon_pos+cannon_speed<=620)
149:                 cannon_pos += cannon_speed;
150:         putcannon(WHITE);                       /* 砲台を表示 */
151:
152:         for(i=0; i<MAX_SHOTS; i++) {
153:                 if(shot[i].stat==0)             /* 飛んでいる弾のみ処理する */
154:                         continue;
155:                 putshot(i, BLACK);
156:                 shot[i].x += shot[i].speed_x;
157:                 shot[i].y += shot[i].speed_y;
158:                 shot[i].speed_y += 1;           /* Y方向は等加速度 */
159:                 if(shot[i].y>=LAND_Y-2)         /* 弾が地面に落ちたか? */
160:                         shot[i].stat = 0;
161:                 else
162:                         putshot(i, YELLOW);
163:         }
164: }
165:
166: void generatebomber(int n)              /* 爆撃機を生成する */
167: {
168:         bomber[n].stat = 1;
169:         bomber[n].bstat = 0;
170:         bomber[n].x = 0;
171:         bomber[n].y = rand() % 200 + 20;        /* 高度を乱数で決める */
172:         bomber[n].speed = 1 + rand() % 2;       /* 速度(1または2) */
173: }
174:
175: int ishit(int n)                /* n番の爆撃機が砲弾に当っているか */
176: {
177:         int i;
178:
179:         for(i=0; i<MAX_SHOTS; i++)
180:                 if(shot[i].stat==1 &&
181:                  abs(shot[i].x - bomber[n].x) <= 15 &&
182:                  abs(shot[i].y - bomber[n].y-5) <= 7)
183:                         return 1;
184:         return 0;
185: }
186:
187: void movebomber(int n)                  /* 爆撃機を動かす */
188: {
189:         if(bomber[n].stat==0 && bomber[n].bstat==0) {
190:                 if((rand()%150)==0)
191:                         generatebomber(n);
192:                 return;
```

ゲームの遊び方

プログラムを起動すると，画面下に地面をあらわす緑の線がひかれ，中央に砲台が表示されます．次に左上から敵機が出現し，画面左から右へ爆弾(水色の弾）を落としながら通り過ぎていきます．

砲台についている砲身はつねに左右に（ワイパーのように）動いていますが，スペースキーを押せばそのとき砲身が向いている方角に迎撃弾（黄色の弾）を撃つことができます．弾は放物線を描いて飛ぶので，動いている敵機にうまくタイミングをあわせて撃ってください．リピートをきかせる（スペースを押し続ける）と弾幕を張ることができます．弾が敵に当たると敵は破壊され，左上に点数が表示されます．

砲台はH（または4）で左へ，L（または6）で右へ動かすことができます．動きながら弾を撃つこともできます．いったん動きはじめると，何かほかのキー（Jや5など）を押すか左右の端に達するまで動き続けます．ある程度以上左へは行けなくなっていますが，これは画面左端が敵弾に対する安全地帯となってしまうためです．

砲台が敵の落とす弾に当たってしまうとゲームオーバーです．途中でゲームをやめたいときはESCキーを押して自爆してください．

同時に出現する敵の数や撃てる弾の数は，プログラムの53，54行目で定義

```
193:         }
194:
195:         if(bomber[n].stat==1) {
196:                 putbomber(n, BLACK);
197:                 if(ishit(n)) {                          /* 弾に当っているか */
198:                         printf("¥x1b[1;1Hscore : %d", ++score);
199:                         bomber[n].stat = 0;
200:                 } else {
201:                         bomber[n].x += bomber[n].speed;
202:                         if(bomber[n].x >= 635)          /* 画面右端まで到達? */
203:                                 bomber[n].stat = 0;
204:                         else
205:                                 putbomber(n, WHITE);
206:                 }
207:         }
208:
209:         if(bomber[n].bstat==0) {
210:                 if(bomber[n].stat==1 && (rand()%100)==0) {
211:                         bomber[n].by = bomber[n].y;     /* 爆弾を生成 */
212:                         bomber[n].bstat = 1;
213:                         bomber[n].bspeed = 1;
214:                         bomber[n].bx = bomber[n].x;
215:                         putbomb(n, CYAN);
216:                 }
217:         } else {                        /* 爆弾を動かす */
218:                 putbomb(n, BLACK);
219:                 bomber[n].bx += 4;
220:                 bomber[n].by += bomber[n].bspeed;
221:                 bomber[n].bspeed += 1;
222:                 if(bomber[n].by>=LAND_Y-10 && abs(bomber[n].bx-cannon_pos)<=15)
223:                         cannon_destroyed = 1;
224:                 if(bomber[n].bx >= 640 || bomber[n].by >= LAND_Y-2)
225:                         bomber[n].bstat = 0;            /* 画面外に出た */
226:                 else
227:                         putbomb(n, CYAN);
228:         }
229: }
230:
231: void game(int wait)                     /* メインルーチン */
232: {
233:         int i;
234:
235:         while(!cannon_destroyed) {      /* 砲台が破壊されるまでループ */
236:                 movecannon();                   /* 砲台の処理 */
237:                 for(i=0; i<MAX_BOMBERS; i++)
238:                         movebomber(i);          /* 爆撃機の処理 */
239:                 for(i=0; i<wait; i++)
240:                         sin(0.5);       /* (時間待ちのための無意味な計算) */
241:         }
242:         for(i=0; i<999; i++) {          /* 破壊された砲台を点滅させる */
243:                 putcannon(WHITE);
244:                 putcannon(RED);
245:         }
246: }
247:
248: void gameinit()                         /* ゲーム初期化 */
249: {
250:         int i;
251:
252:         setcolor(GREEN);
253:         moveto(0, LAND_Y);
254:         lineto(639, LAND_Y);
255:         cannon_pos = 320;
256:         cannon_speed = 0;
257:         cannon_dir = 0.5;
258:         cannon_dd = 0.03;
259:         cannon_dx = cos(cannon_dir)*CANNON_SIZE;
260:         cannon_dy = sin(cannon_dir)*CANNON_SIZE;
261:         for(i=0; i<MAX_BOMBERS; i++) {
262:                 bomber[i].stat = 0;
263:                 bomber[i].bstat = 0;
264:         }
265:         for(i=0; i<MAX_SHOTS; i++)
266:                 shot[i].stat = 0;
267: }
268:
269: int main(int ac, char *av[])
270: {
271:         time_t seedtime;
272:
273:         if(ac!=2) {                             /* パラメータ数をチェック */
274:                 printf("usage: SHOOT {wait_time}", av[0]);
275:                 exit(1);
276:         }
277:
278:         time(&seedtime);
279:         srand((unsigned int)seedtime);          /* ランダマイズ処理 */
280:
281:         graphinit();                            /* 画面モードの初期化 */
282:
283:         gameinit();                             /* ゲームの初期化 */
284:         game(atoi(av[1]));                      /* メインルーチン */
285:
286:         while(kbhit())
287:                 getch();
288:         getch();
289:
290:         graphdeinit();                          /* 画面モード復帰 */
291: }
```

しています．この値を変更すれば，敵の数を増やして難易度を上げたり，弾切れをなくしたりすることができます．

プログラムの説明

8-49行

Turbo CとMS-C（またはQuick-C）のグラフィックス関数の違いを吸収するための定義など．

51-54行

各種定数の定義．

56-64行

砲弾および爆撃機（爆弾を含む）の状態を保存する変数の定義．それぞれ複数の変数からなっているので，構造体を使ってまとめて定義します．

66-70行

砲台の状態を表す変数と得点の変数の宣言．

putbomber()

爆撃機を指定された色で画面に表示します．黒色を指定することによって，表示した爆撃機を画面から消去するルーチンもかねています．

putcannon()

putshot()

putbomb()

それぞれputbomber()と同様に砲台，（砲台から撃つ）砲弾，（爆撃機が落

とす）爆弾を画面に表示します．

shoot()

砲台から弾を発射します．まず弾データ(shot[n])の中で使用されていないものを捜し，すべて使用中ならなにもしないで戻ります(弾切れ状態)．空きが見つかれば，そのときの砲台の位置と砲身の方向から弾データの初期値を設定します．

movecannon()

砲台を動かす処理をします．

121–141行

キー入力があれば，入力されたキーに応じた処理をします．

142–150行

表示されている砲台をいったん消去し，砲身の向きと砲台の位置のデータを更新してから再度砲台を表示します．

152–163行

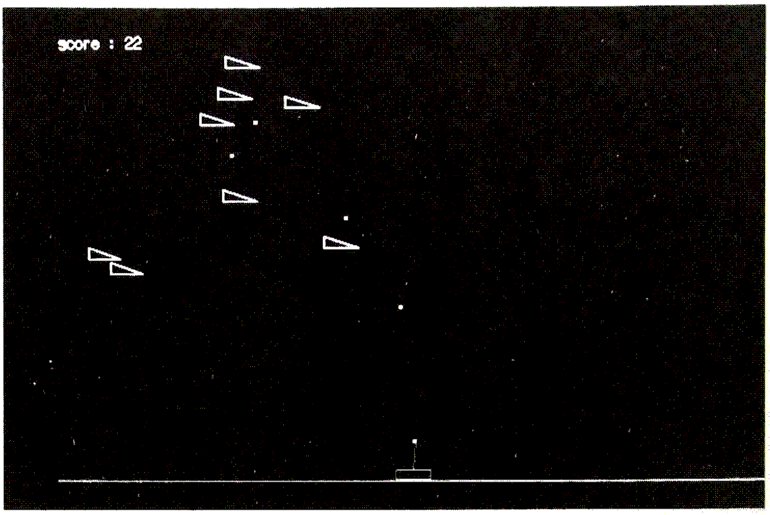
弾を動かします．すべての飛んでいる（shot [n].stat==1）弾について，表示されている弾を消去してから位置データを更新し，地面に落ちたならデータを解放（shot [n].stat=0）し，まだ飛んでいれば新しい位置に表示します．

generatebomber()

爆撃機を新たに生成します．171行目で爆撃機の高度をY座標20～219の範囲でランダムに設定しています．その次の行で爆撃機の速度を乱数で1か2にしています．

ishit()

指定した爆撃機が砲弾に当たっていれば1を，当たっていなければ0を返す関数です．Y座標については砲弾と爆撃機が完全に重なっていなくても，



ある程度以上近ければ当たっていると判定しています（ワンポイントテクニックの項を参照)．

movebomber()

指定された爆撃機を動かす処理をします．

189–193行

指定された番号のデータが空きデータのときは，1/150の確率で新たな爆撃機を生成します．

195–207行

爆撃機が飛行中であれば，表示されている爆撃機を消去し，砲弾に当たっていれば得点を加算して表示し，そうでなければ位置を更新し，まだ画面内にあれば新たな位置に爆撃機を表示します．

209–216行

爆撃機が飛行中で爆弾が投下中でなければ1/100の確率で新たに爆弾を生成します．

217–228行

爆弾が投下中であれば表示されている爆弾を消去し，位置を更新し，砲台に当たっていればフラグを立て（cannon_destroyed=1)，画面内にあれば新たな位置に爆弾を表示します．

game()

ゲームのメインループです．砲台が破壊されるまでmovecannon()とmovebomber(n)を順に呼び出し，ループ毎に時間待ちを行います．砲台が破壊されてループを抜けてきたら赤と白で点滅させて破壊をあらわします．

gameinit()

ゲーム開始にあたって，変数の初期化などを行います．緑色で地平線をひき，砲台の初期位置を設定して，爆撃機と砲弾の各変数の状態をクリアします．

main()

パラメータ数が不正ならばプログラムの起動方法を簡単に表示して終了します．乱数を初期化し，画面の初期化，ゲーム初期化，ゲームのメインループを順に呼び出したあとキー入力待ちをしてから画面を復帰してプログラムを終了します．

ワンポイントテクニック

アクションゲームの基本構造

今回のようなアクションゲーム（画面内をキャラクタが動き回るゲーム）のプログラムは，多くの場合共通した構造をもっています．複数のキャラクタが同時に動いているように見せかけるため，リスト2のようにメインループの中で順に各キャラクタの移動ルーチンを呼ぶのがその基本的な構造です．このキャラクタの移動ルーチンにはキャラクタの生成や消去，当たり判定などの処理も含まれていますが，核となるのは実際に画面上でキャラクタを移動させる部分です．

キャラクタの移動は，プログラム的には3つの段階に分かれていると考えられます．すなわち「画面からの消去」「位置データの更新」「画面への表示」の3つです．移動といっても実際には画面上の1点から1点へ飛び移っているのですが，人間の目には連続して移動しているように見えるわけです．消去してから表示するまでに時間がかかるとキャラクタが点滅して見えてしまうので，データの更新に時間がかかる場合にはもとの位置データを残しておいて，消去と表示を続けて処理したほうがよいでしょう．

位置データを更新するときにはいろいろな判断（キャラクタが移動範囲を越えていないかなど）も行いますが，ゲームにおいてもっとも重要なのが「当たり判定」（キャラクタ同士の衝突判定）です．敵に弾が当たったかどうか，プレイヤー（今回は砲台）に敵の弾が当たったかどうかなどが正確に判定されないとゲームとして成立しなくなってしまいます．しかし，多くのキャラクタが同時に画面に表示されていると，当たり判定を行う回数もそれだけ増えるため，できるだけ単純な計算で判断する必要があります．短い時間で判定する方法は多様ですが，今回のプログラムではキャラクタ同士の中心の位置を比べてXY方向の差分をとり，その数値の大小によって当たり判定を行っています．つまり，敵機は画面上では三角形ですが，長方形とみなして判定しているわけです．

通常はこの方法で十分ですが，今回の場合1つ大きな問題がありました．砲台や敵機自体はゆっくり（各ターンごとに数ドット）動いているので問題ないのですが，弾はかなりとびとびに動いています．人間の目は間を補ってくれるのできれいに放物線を描いて飛んでいるように見えますが，弾のスピードが速いところ（下のほう）では20ドット以上の間隔があいているのです．ちょうどその中間を敵機が通過すると，人間の目では弾が敵機をすり抜けているように見えます．このため，Y方向は敵機の大きさより少し大きめに当たり判定をとって，できるだけ敵機がすり抜けてしまわないようにしました(182行目)．あまり大きくとると，こんどは当たっているように見えない敵機が破壊されてしまうことになるので加減が必要です．今回はこれで十分だと思いますが，遊んでみて不都合を感じるようなら，もっと正確な判定方法を考えなければいけません．たとえば今の場合では弾のY方向の速度(shot [n].by)の大きさに応じて当たり判定の大きさも変えるなどの方法です．さらに正確には，弾の軌跡を計算してキャラクタとの交点を求めるということもできますが，時間がかかりすぎてあまり実用的ではないでしょう．

今回のプログラムは行数の関係もあって細かい部分に手抜きが多く，クリッピング（キャラクタと画面の境界との重なり）の判定をさぼっている（関数まかせになっている）とか，手順が不正確で表示していないキャラクタを消していたりと，気もち悪い部分が残っていますが，ゲーム進行上とくに問題とはならないので放ってあります．このようなプログラムは細かな部分や処理の手順に結構神経を使うものなので，全体の構造が単純な割にめんどうなものだと思います．

ゲームのスピードを調整するため，メインループに入れる時間待ちも重要ですが，今回は簡単にするためプログラムの起動時に適当な待ち時間の値を指定してもらうようにしています．本来なら機種による速度の差を吸収し，ループの各ターン毎の時間間隔を一定にするため，割り込みルーチンなどを使って時間待ちを行うべきです．割り込みを使った時間待ちについては，次回以降で取り上げたいと思います．

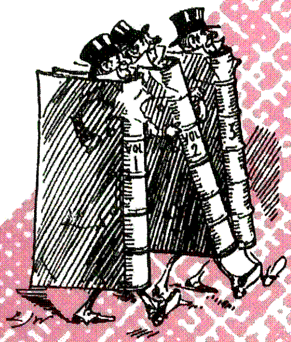
リスト2

```
while(ゲーム終了フラグ==0) {
        キャラクター1の移動ルーチン
        キャラクター2の移動ルーチン
                  :
                  :
}
```

IC千夜一夜

第3回 8ビットマイクロプロセッサの発展

—8008と8080—



虎岩 登

世界初の8ビットマイクロプロセッサ

8008は4004の直系

先月の4004と4040の比較に続き，今月はintel社の8008と8080を比較してマイクロプロセッサの発展史を見ていきましょう．8008は世界初の8ビットマイクロプロセッサであり，1972年に発売されています．先行する4004と同じp-MOSプロセスで作られている18ピンのICです．ピン配置は図1を見てください．本当にシンプルなものです．

この8008は，前年に発表した4004が評判になって，同じアイデアでコンピュータの文字端末用に使えるマイクロプロセッサが欲しいという注文に応えて設計されました．ところがハードウェア構成や動作時のタイミングまで4004と似せてしまい，それをサポートするICはほとんど発表されませんでした．そのために，非常に利用するのが難しい，外付け回路の多い不便なマイクロプロセッサという評価を受けることになります．なにしろ当時の集積度のICでは，8008を用いたCPU部分は(コンピュータ全体ではありません)TTL ICを30～50個ほど外付けしないと構成できませんでした．

ピン配置を注意深く眺めてみると，リセット信号がありません．リセットするには，割り込みをかけて強制的にRST命令（リスタート命令，ある種のソフトウェア割り込み命令）を読み込ませて，特定の番地へジャンプさせるという技を使います．パワーオン・リセット用の回路も結構めんどうなのです．

動作も4004と同じく遅さが目だちます．クロックが500KHzで，1命令実行の最短時間は20μsも必要でした．スタックも4004と同じくメモリ上に割り当てられずに，チップ上に7つ分だけ用意されています．このスタックはサブルーチンコールのリターンアドレス格納専用でレジスタをpush/popすることはできませんし，当然のことですがサブルーチンコールは7レベルの深さまでしか許されません．8レベル以上のサブルーチンコールを誤って行うと暴走するというきつい制限がありました．事実上，再帰呼び出しなど許されてはいません．ただし，4004と異なり，メモリ空間が最初からROMとRAMに分かれておらず，現代風に同じメモリ空間にプログラムとデータが同居する構成になりました．

プログラマから見えるレジスタ構成を図2に示します．これも先月の4004のレジスタセットと比較するとおもし

ろいと思います．4004でスクラッチパッドレジスタと呼ばれていたものがそのまま存在しています．ただし，数は約半分に減ってしまって（ビット数では同じくらい？），名前が数字でなくてアルファベット1文字に変わっています．B，C，D，E，H，Lですね．アキュムレータは相変わらず演算命令で特別に扱われているのですが，頭文字をとってAレジスタと呼ばれて，あたかもスクラッチパッドレジスタの一部のように表記されます．BからEまではしゃれで付けたようなレジスタ名です．HとLレジスタがなぜF，Gでないのかといえば，ある種の特権的な機能があったからです．つまりHとLを組み合わせてポインタとして利用できたのです．Hはアドレスの上位(High)，Lはアドレスの下位（Low）を入れるという意味でしょう．HLレジスタペアの内容をメモリのアドレスと解釈して，そのメモリに対してほかのスクラッチパッドレジスタと同じような転送命令や演算命令が利用可能でした．アセンブラの表記法では，HLレジスタペアの指すメモリ内容を，これまたしゃれてMレジスタと呼ぶ決まりになっていました．

表1に8008の命令セットをひと通り簡単にまとめたものを載せておきます．後述しますが，8080のアセンブラのニーモニック表記と同じですから，おわかりの方も多いでしょう．

図1　8008，8080のピン配置

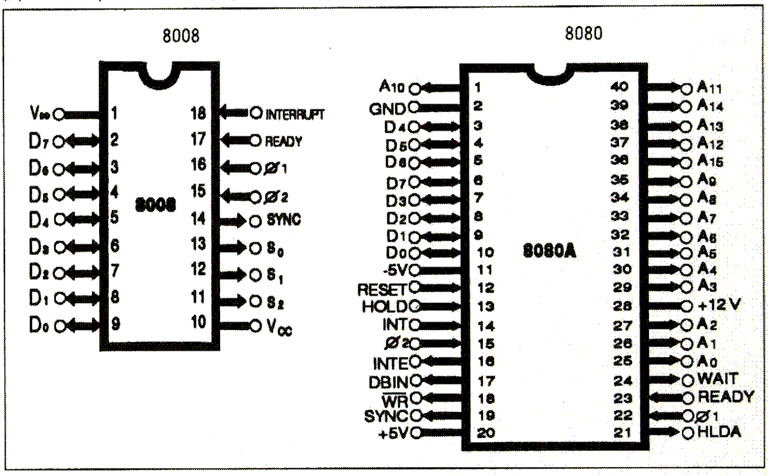
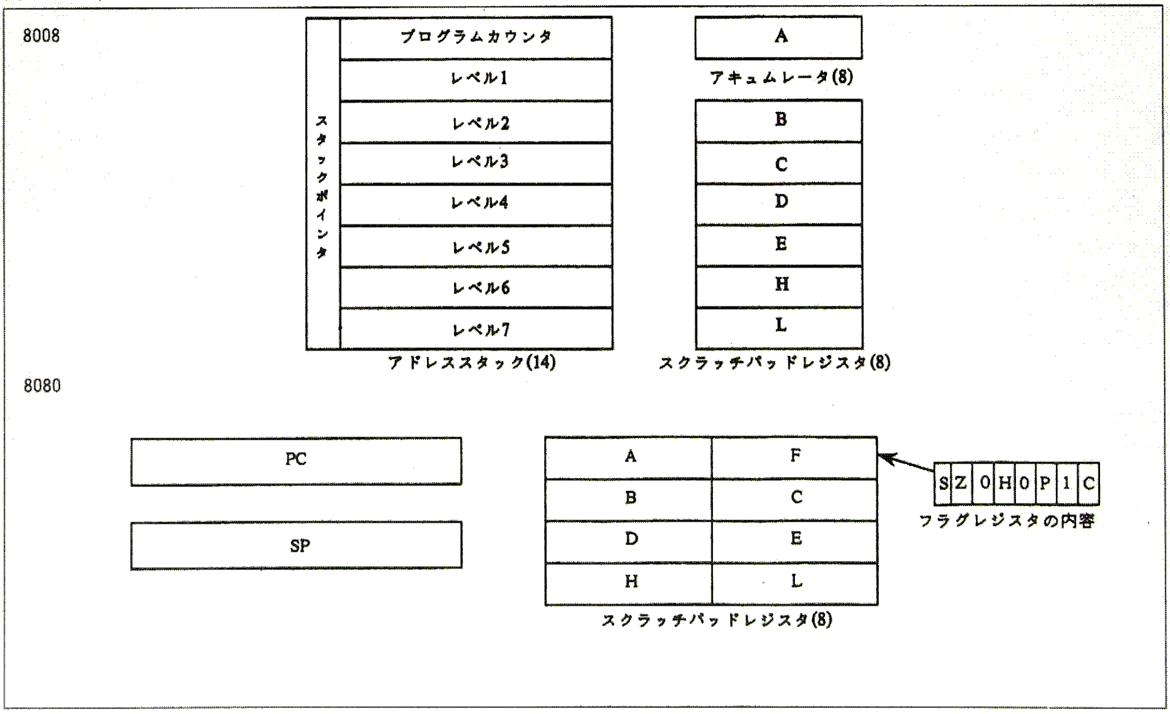


図2　8008，8080のレジスタ構成



新世代マイクロプロセッサ

8080の登場

8008の使いにくさを大幅に改良したのが，1974年に発表された8080です．4004や8008で１組のバスを時分割で切り替えてアドレスやデータの入出力に使っていたのをやめて，アドレスバス・データバスを別々の端子に割り当てています．そのため，ピン数が大幅に増え，40ピンのパッケージが使われました．また，n-MOSプロセスを利用し，クロック周波数を2MHzまで引き上げ，最短命令実行時間は2μsにまで高速化しました．まだ外付け回路は多少は必要ですが，一般的なメモリICへの接続も容易でした．ただし，8080は+5V，+12V，-5Vの３電源が必要であり，入力電圧が厳密にはTTLレベルではなかったり，いくつかの欠点は残っています．しかし，ユーザーの立場からは，新しい世代のマイクロプロセッサの登場といえる，画期的なできごとでした．

とはいえ，上にあげた点はハードウェア面から眺めたものです．プログラマから見たモデルとしては，スタックポインタが付け加わり，スタックをメモリ上に取れるようになったこと，フラグ類がフラグレジスタＦとしてまとめられ，Ａレジスタのそばに書かれていることが違うだけで，レジスタモデルはほとんど変わっていません．また，8008と8080は，バイナリレベルではコンパチブルではないけれど，アセンブラレベルでは上位コンパチブルになっていました．CPUが解釈する命令のビットパターンは同じではないのですが，アセンブラのソースコードでは，8008に存在した命令はすべて8080に含まれています．そして，さらにプログラムを書く上で重要な命令がたくさん追加されています．それらを比較してみましょう．

表１　8008の命令のニーモニック表記のすべて

■スクラッチパッドレジスタ命令				
MOV	MVI			転送命令
INR	DCR			インクリメント・デクリメント
■演算命令				
ADD	ADI			加算
ADC	ACI			加算（キャリー付き）
SUB	SUI			減算
SBB	SBI			減算（キャリー付き）
ANA	ANI			論理積
XRA	XRI			排他的論理和
ORA	ORI			論理和
CMP	CPI			比較
RLC	RRC	RAL	RAR	ローテート（シフト）
■ジャンプ・コール・リターン命令				
JMP				無条件ジャンプ
JNC	JNZ	JP	JPO	条件付きジャンプ
JC	JZ	JM	JPE	
CALL				無条件サブルーチンコール
CNC	CNZ	CP	CPO	条件付きサブルーチンコール
CC	CZ	CM	CPE	
RET				リターン
RNC	RNZ	RP	RPO	条件付きリターン
RC	RZ	RM	RPE	
RST				リスタート命令
■入出力命令				
IN	OUT			
■その他				
HLT				CPUの停止

8008と8080の違いは，あくまで8008はデータを８ビット単位でしか処理できなかったことにあります．この差は演算命令の16ビット幅加算DAD D（ADD HL，DE　注：カッコ内はZilog表記，以下同じ）などがないというだけの違いではありません．HLレジスタペアに16ビット値をイミディエートでロードすることもできませんでした．つまり，

```
LXI H,0100H (LD HL, 0100H)
```

というような命令は8008には存在しませんでした．この場合は8008では，

```
MVI H, 01H
MVI L, 00H
```

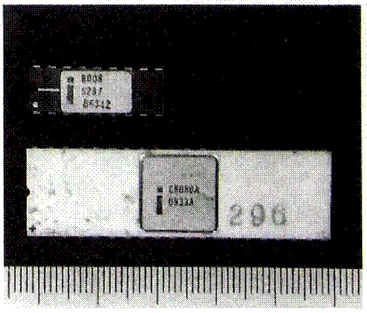
と別々にロードするのです．また，メモリ空間の特定のアドレスからデータを直接読み取る命令も8080になってはじめて追加されました．1234H番地の内容をＡレジスタに読み込みたくても，

```
LDA 1234H (LD A, (1234H))
```

のような命令はなく，まずHLレジスタペアに1234Hをロードして，

```
MOV A, M
```

写真１　8008（上），8080（下）．



という手順を踏みます．8080ですら1命令6.5μsで実行可能なこの操作に，8008では3命令96μsも必要になります．しかもこのときにはHLレジスタペアの前の値は壊されているので，別の番地のメモリにアクセスしたいときには再びHLレジスタペアに値をロードしなければいけません．

たとえば，メモリ内のデータを移動するようなサブルーチンを考えてみましょう．8080では，BC，DEレジスタペアもポインタとして利用する命令が追加されていますから，DEレジスタペアとHLレジスタペアの2つのポインタを使ってAレジスタを仲介してコピーできます．そうすると転送残りバイト数はCレジスタでも使ってカウントしてループすることができますね．

ところが8008では，ポインタとして使えるのはあくまでHLレジスタペアだけですから，BCレジスタとDEレジスタを転送元と転送先のメモリアドレスを記憶するのに使い，交互にHLレジスタペアにもってきてデータ転送しなければいけません．そうすると，レジスタを使いきってしまうので，ループカウンタは適当なメモリ上にワークエリアとして確保しておいて，これまたHLレジスタペアでポイントして参照することになります．しかも16ビットのレジスタペアに対するインクリメント命令もないので，転送範囲が256バイト境界にまたがってしまうとBCレジスタの内容をインクリメントするのも大騒ぎです．もちろん，転送先のアドレスやバイト数が最初から固定されているときは，転送先アドレスが特定の値になったかどうかをイミディエート・アドレッシングの比較命令で検査してループから抜け出すことが可能ですから多少は簡単になります．

8008が何かほかの機器の内部の組み込み専用機として遅い処理でも許されるような場合にしか使われず，普通の意味でのパーソナルなコンピュータとしては使われなかった理由が，なんとなくわかってもらえたでしょうか．ちょっと複雑なことを行おうとすると，本当にパズル的な膨大なプログラムになってしまうのです．

一方8080では，レジスタモデルや基本的な命令体系はほとんど変更せずに，強力な命令の追加という手法で，より大規模で高速な処理が要求される応用に適した改良が行われました．当時のハードウェア技術者が片手間にソフトも作成するという状況では，当然，ハードウェア面が使いやすくなったことも応用の広がりへのはずみになりました．それらが相まって，マイクロプロセッサの広範な分野への普及がはじまったのです．

今回は，付録ディスクに特別におまけを付けることにします．絶対に役に立たないこと請け合いのプログラムですが，詳しくはディスクの中身を見てください．

ICパッケージ――DIP

ICの中身であるチップは，保護のため，あるいは実装を容易に行うため，さまざまな形状や材質の容器に封入されています．今回は昔ながらのDIPと呼ばれるパッケージの話です．

DIPとはDual In-line Package，端子が2列に並んだパッケージという意味で，「ディップ」と読みます．今回の8008や8080（写真1）のように，パッケージの両側から端子が出ていて，コの字型に曲げられているタイプをいいます．一時期はデジタルICのパッケージはこれしかないというほど広く使われていました．製造が比較的容易で，昔ながらのプリント基板に取り付けるのに適していたこと，組み立てラインの自動化にも適している（自動挿入・自動ハンダ）などの点から広く使われました．

初期のころは14ピンと16ピンが主だったのですが，オペアンプなどの8ピンや，メモリ・マイクロプロセッサなど多数のピンが必要なICが増えるにしたがって18，20，22，24，28，32，40，42，48，64ピンなど多くの種類が使われるようになりました．

今では，さらに多ピンのICに対する要求や，より小型化，表面実装などの特殊な（現在ではもはや特殊とはいえないかもしれません）実装法に適したパッケージの要求が強まり，DIPパッケージ以外のものが使われる機会が増えています．

基本的にはICのために設計されたパッケージですが，同じようなパッケージにすると実装コストが安くなったりするので，スイッチやリレー，抵抗アレーなども同様の形状のものが作られています．コンピュータのマニュアルなどで見かける「ディップスイッチ」というのも，本来はDIP型の端子配列をしたスイッチという意味です．

MS-DOS TOOL LABO

MS-DOSツール・ラボ

第6回 コンパイラの自己記述 FAST

FORTHコンパイラ②

加野島英渡

前回に引き続いて，スタックを使う独特の言語FORTHのコンパイラであるFAST (Forth Assembler Sourcecode Translater)の作成を行う．今回は，FASTコンパイラ自身をFASTで記述し，さらにFASTの文法について詳しく解説する．

FASTをFASTで記述する

このコンパイラはこのコンパイラ自身で記述されています

あたりまえになってしまったのか最近さすがにこの気恥ずかしいコピーは見なくなったが，Cなどのシステム記述言語を標榜するコンパイラのひと昔前の広告にはこれが誇らしげに踊っていたものだ．今回ご紹介するFASTもFAST自身で記述されている（あ，恥ずかしいなどといいながら書いてしまったぞ）．

コンパイラというプログラムはそこそこに複雑であって，データ構造や複雑な制御構造など，それを記述する言語をごりごりと使うことになる．だからその記述言語として自分自身を使用できるということは，それがそういった高度なシステム記述をするだけのパワーがあるということの証明になるということのようだ．

ところが，これはあまりあてにならない．いやしくも汎用言語であれば得手不得手はあるものの，たいがいのことには使えるので，自分自身を記述できるからといって必ずしもその言語が優秀でシステム記述に向いているとはいえない．別にBASICの悪口をいうつもりはないのだが，その昔BASICで書かれたBASICコンパイラなんてすごいものまであったくらいだもの．

ところで，コンパイラをその言語自身で記述するというのはいいが，その言語はどこから来たのだろう？

たとえば「CはC自身で書かれている」というが，最初にCコンパイラを書いたときにはCコンパイラというのはなかったはずだ．いったいどうやって書いたんだ？

わかってみれば簡単なことで，「最初

ワンラインNEWS
野村総合研究所がまとめた「90年代の半導体産業」と題する調査結果では，将来フラッシュメモリが磁気メモリにかわるだろうと予測している．

はそれ以外の既存の言語で書いた」のである．

■コンパイラ自己記述の手順

①まず既存の言語の中からシステム記述に向いた言語Aを選び出し，この言語Aでこれから作ろうとする新しい言語，Bコンパイラの基本的な部分を記述する．
②そのソースを言語Aのコンパイラでオブジェクトに変換する．
③Aで書かれたBコンパイラのオブジェクトができたら，言語BでBコンパイラのソースを記述しなおす．
④そのソースをBコンパイラでコンパイルすると，言語Bで記述したBコンパイラのオブジェクトができる．
⑤言語Bで書かれたBコンパイラのソースを機能拡張し，④へ

という具合で，①から③の作業は最初の1回だけで，あとは自分で自分自身の機能拡張ができるようになる．これではちょっとわかりにくいので，具体的な言語名をあてはめてみよう．これはそのまま今回のFASTコンパイラの開発の経過でもある．

①まずCの文法でFASTコンパイラの基本的なバージョンを記述する．これをFASTCと名づける．
②これをコンパイル（もちろんCコンパイラで）してFASTCコンパイラのオブジェクトを作る．
③FASTの文法でライブラリを記述し，また，いくつかのテストプログラムを記述する．
④コード生成を確認して，バグがあったら(必ずあるのだよ)コンパイラのソースを修正して②へ．
⑤FASTCコンパイラのコード生成とライブラリの信頼性が確認できたら，Cの文法で書かれたFASTCコンパイラのソースをFASTの文法で記述しなおす．
⑥FASTの文法で記述されたFASTコンパイラのソースをFASTCでコンパイルしてライブラリとリンクし，FASTで記述されたFASTコンパイラのオブジェクトを作る．これをFASTFと名づける．
⑦FASTFでFASTコンパイラのソースをコンパイルしてライブラリとリンクし，FASTFとおなじオブジェクトが生成されていることを確認する．
⑧コンパイラの機能を改良する．
⑨バグや機能不足があったらソースを修正して⑧へ．

少し頭が痛くなってきたかな？　開発段階ではどれがどのバージョンのコンパイラで作られたコンパイラかがわからなくなってしまうので，ファイル名に変化をつけたり適宜ディレクトリを移したりして混乱を防ぐ工夫が必要だ．

①から④の段階，つまり既存の言語でコンパイラを記述しているときには，最低限の機能の実現を目標とし，思いつきの機能拡張などは厳に慎むべきである．作っているうちにどんどんアイデアが出てくるものなのだが，ここでそれに入ってしまうとあとで自己記述に移るときの手間が増える一方なので，じっと我慢しなければならない．既存の言語としては自分がふだんよく使っているものを選ぶことが普通だから，気がついたらソースが巨大になってしまって身動きがとれないという悲劇も考えられる．

自己記述段階に移ったら，好きなだけ機能拡張していけばいい．この段階で注意しなければならないのは，コンパイラのソースを修正して新しいコンパイラを作ったときである．このときはまだ新しいコンパイラの機能は確認されていないので，非常に不安定な状態にある．まちがってもこの不安定なコンパイラをよく検査せずに使用してはならない．この段階のコンパイラで自分自身のソースをコンパイルしてしまったら，いったいなにが出てくるか見当もつかない．

検査のためには，自己記述前の段階からテストプログラムをどんどん作ってためておくことがいちばん有効である．テストプログラムごとに信頼性の高いバージョンのコンパイラのコード生成を保存しておき，新しいコンパイラができたらすべてのテストプログラムをとおしてコード生成を比較する．

コード生成に関わる変更をしたときは要注意である．このときは人間の目で追うしか方法がない．コンパイラの機能拡張というのは生成コードの品質に関わることが多いので，なかなかやっかいなものである．

ところで自己記述というのはコンパイラだけではなく，たとえばエディタでもこういうことができる．使い勝手の悪いエディタしかない場合，最初にその使い勝手の悪いエディタで使い勝手のいいエディタのソースを書く．それをコンパイルなりアセンブルなりすれば，使い勝手のいいエディタのファーストバージョンができあがる．あとはそのエディタでさらに使い勝手のいいエディタのソースを編集していけばいいのである．

こういうのをブートストラッピングと呼ぶそうだ．自分で自分の靴紐を引っ張り上げるという意味だと聞いたが，外人とはけったいなことを考えるものだなぁ．

FASTの実際

■FASTの特徴

FASTは，テキストファイルの中から

FASTの文法にしたがって記述された部分をアセンブラ(MASM)のソースに変換する1パスのFORTH系コンパイラである．FAST記述以外の部分にはいっさい介入せず，素通りさせる．

極端な話だが，ドキュメントファイルのようなものであっても，その中にFAST記述部分があれば，そこをMASMのソースに変換して出力する．それ以外の部分にはまったく手をつけない．オブジェクト（FASTではMASMのソース形式に変換されたテキストを「オブジェクト」と呼ぶ）の出力は標準出力に対して行われるので，これをリダイレクトすることでファイルに落とすことができる．実際には，簡単なバッチファイルを使って作業を行う．ソースファイルを標準入力から与えることもできるので，フィルタのような使い方も可能である．

出力されるMASM形式のソースにはファイル名とソース行番号，ソースイメージを1行にまとめてコメントの形で入れるので，アセンブルでエラーが出た場合に，エラー出力→アセンブラソースファイル→FASTソースファイルというふうに2回のタグジャンプでソースファイルの該当箇所にたどり着くことができる（標準入力からソースを与えた場合はソースファイル名は出力されない）．

起動時は素通しモードになっており（これを仮にアセンブラモードと呼ぶ），FAST記述の開始を認識するとFASTモードになる．FASTモードの中でFASTモードの終了を意味するワードを認識すると，またアセンブラモードに戻る．

生成されるオブジェクトはMASMのソースであるから，これをアセンブル・リンクすることでMS-DOS上で実行可能なファイルを作ることができる．ライブラリとのリンクも自由だし，もし必要ならばFASTで記述されたプログラム（ワード）をライブラリ化することも可能だろう．セグメント定義などはすべてプログラマの手に委ねられているので，プログラム作成の自由度は非常に大きい．

FAST記述部分に適宜アセンブラのソースコードを埋めこむことができ，ちょうどインラインアセンブラのような使い方が可能である．コア（基本ライブラリ）はほとんどこの方法で記述されている．

FASTは汎用言語として設計されているが，

・コンパイラやエディタ，デバイスドライバや常駐ソフトといった従来アセンブラやCなどで書かれていたシステムプログラム．
・機械制御や計測といった本来FORTH系の言語が得意としているような分野．

といった方面にとくにその適性を発揮する．もちろん一般のアプリケーションの記述も可能である．

なお，先月号でFORTHのオブジェクトは実行時インタプリタによって実行されると書いたが，FASTは実行時インタプリタが介入しないネイティブコードコンパイラである．

■FASTの文法

FASTはFORTH系の言語であり，文法はFORTHに準ずる．変数や定数記述などの機能は用意されておらず，これらはアセンブラの文法に準ずる．もちろんFAST記述の部分からアセンブラで記述されている定数や変数へのアクセスが可能である．

FASTでは，ワードはその名前を単に記述することで参照（呼び出し）される．したがって，ワード名をただ並べていくことで処理を記述することができ，あるまとまった処理手順にそれらを代表する名前をつけることで，新たなワードを作成することができる．この考え方は一般の手続き型言語となんら変わるところはない．強いていえば，引数や戻り値が暗黙のうちにスタックを相手にやり取りされるため，「そのワードに対する引数」という明確な記述方式をもたないという程度のことである．

■名前の認識

FASTはスペースとTABを名前の区切りとし，これらではさまれた文字列を名前として認識する．この名前をコンパイラ内部では「ワード名」と呼ぶ．

■名前の種別

(1) 数値

- **ワード名に含まれる文字がすべて十進数値であるもの．**
- **ワード名の末尾が'H'または'h'であり，それ以外の名前に含まれる文字がすべて十六進数値であるもの．**

- **'(' で始まり')'で終わるワード名(現在のバージョンではカッコのネストは許されていない)．**

これは定数宣言を参照するためである．FASTはアセンブラモードの記述にはいっさい関与しないため，数値以外のものが定数であるのかワード名であるのか判断できない．ただ名前を書くとその名前を呼び出すようなコードを生成してしまうため，この記号によってこれが数値であることを知らせ，特別なコード生成を指示する．定数式や負数もこの中に記述する．

(2) アドレス（オフセット）

- **名前の先頭が'#'であるものをアドレス（オフセット）を表すものと認**

ワンラインNEWS
国内のコンピュータ大手8社の90年度決算が発表された．各社10%前後の伸びを見せる中で日本IBMは1.9%と伸び悩み，日本電気に抜かれ3位となった．

識する．

FASTはアセンブラモードの記述にはいっさい関与しないため，名前が変数であるのかワード名であるのかが判断できない．ただ名前を書くとその名前を呼び出すようなコードを生成してしまうため，この記号によってこれがアドレス(オフセット)であることを知らせ，特別なコード生成を指示する．

(3)　分岐ワード

- "IF"～"ELSE"～"THEN"，"BEGIN"～"UNTIL"／"REPEAT"，"DO"～"LOOP"

(4)　FAST記述の開始と終了

- ソース行の先頭に書かれた'['をFAST記述の開始とし，FAST記述内の任意の位置に書かれた"]"をFAST記述の終了とする．

厳密にいえば，'['を発見した時点ではFASTはアセンブラモード（素通し）になっているので，これは単なる文字として認識され，反対に"]"はFASTモードなので名前として認識されている．シングルクォートとダブルクォートを使い分けたのはそういう理由である．

(5)　一般ワード

- 上記以外のすべて．

■ラベル名への変換

FASTでは，最終的に実行可能なオブジェクトを生成するのにアセンブラの力を借りるので，アセンブラの予約語に相当する名前をワード名として使用できない．しかし，FORTH系言語には"DUP"のようなアセンブラの予約語に相当する基本ワードがいくつかある．このぶつかり合いを避けるために，一般ワードはすべて先頭に"f_"という文字列を強制的につける．

また，FORTH系言語では"＋"や"／"などの演算子も，単独でなければワード名の中で使用することを許しているため，そのキャラクタコードを十六進表現し，前後にアンダースコアをつけることでラベルとして使用できるように変換する．たとえば"IN/OUT"というワード名は，"f_IN_2F_OUT"

COLUMN　FASTをより発展させるには

今回2回に分けて紹介したものは，コンパイラやその作り方ではなく，プログラムの書き方，サブルーチンの管理のしかたにFORTHの思想を使うという考え方である．FASTはその1つのサンプルにすぎない．たとえば，80386以上のCPU専用に32ビット版のライブラリを作ったり，データ構造の定義にまで踏み込んでみたりと，応用はいくらでも考えられる．

今回はコンパイラ自身も含めて全ソースリストを公開したので，ぜひチャレンジしてみていただきたい．"$$startup"の自動呼び出しなどはおもしろいテーマになると思うし，筆者はこれになんとかしてローカル変数をもち込めないか考えている．未整備のコアについても，高度なファイルアクセスワードやウインドウ操作ワードを作成することで，本格的なアプリケーションの作成にも使用することができるだろう．

オブジェクト効率について触れておくが，FASTで記述したFASTコンパイラのサイズが8KB弱である．C（LSI C-86）で記述したこれより機能の低いバージョンのサイズが9KB弱であったので，数字だけを見ると勝っているようだが，コアには必要最低限のライブラリしか用意していないし，一行入力など機能制限をして複数のハンドルが扱えない状態だから，その分を割り引いて考えなければならない．

スピードについてはネイティブコードコンパイラなのでけっして遅くはないが，もともと80x86というCPUがあまりスタック操作が得意ではないことと，演算などもすべてワード呼び出しで行うというFORTH系言語の特徴からあまり速いともいえない．

FASTプログラムの実行速度を上げるには，アセンブラで書くことである……「なんのこっちゃ」であるが，詳しくいうと，シフトなどの細かなビット操作を行う処理，とくにそれがループに入っているような場合は，できるだけ独立した機能にしてアセンブラで書いたほうがスピードが速くなる．FAST記述は，ロジックの記述や速度にあまり影響しないようなところで楽をするために使うというのがコツである．

さて，FORTHという言語は単純ではあるが，そのすべてを説明するにはやはりかなりのページ数を必要とする．

たとえば今回のようなアセンブラのソースジェネレータとして特化したものではない純正FORTHには，コンパイラとしての側面のほかにコンパイル即実行という形の対話型システムという側面もある．これがこの言語を習得するのにどれだけ強力な環境を提供するか，また，開発効率に与える影響がどれほど大きなものか．

また，純正FORTHではIF～ELSE～THENのような制御構造や変数や構造体のようなデータ構造すらプログラマがFORTHレベルで自由に定義できるという高度な自己記述性がある．

FASTはこれらの特性をすべて切り捨てることでFASTとなり得たのである．本来ならばこれらについても説明しておくべきであろうが，ここではそのようなFORTHという言語の奥の深さ，魔力といってもいいような底知れぬ魅力といったものは説明するだけの余裕がない．まことに残念ではあるが，またの機会ということにしたい．

というラベル名に変換される．これによって“DUP”も“f_DUP”となり，アセンブラの予約語とのぶつかり合いが避けられる．このようにして変換されたワード名を「ラベル名」と呼ぶ．

■分岐ワード解説

分岐ワードとは，処理の流れを変更するための各種ワードの総称である．一般のワードがそれを呼び出すための命令に単純に変換されるのに対し，分岐ワードではその分岐を実現するための一連の命令シーケンスが生成される．

“IF”～“ELSE”～“THEN”

“IF”は，スタックトップのデータの真偽を検査する機能である．真であり，かつ“ELSE”があれば“ELSE”まで，“ELSE”がなければ“THEN”までを実行する．スタックトップが偽であり，かつ“ELSE”があれば“ELSE”から“THEN”までを実行し，なければ“THEN”の次に実行を移す．ここで，“IF”が真偽検査のためにスタック上のデータを1つ消費することを忘れないように．

通常の言語では“IF”のあとに条件式を書くが，FORTH系の言語であるFASTでは真偽がスタックに積まれてから“IF”で判定をするのでまちがえないように注意しなければならない．また，“ELSE”と“THEN”の位置も逆なのでこの点も注意が必要だ．“THEN”は“IF”と“ELSE”の合流点であり，意味合いとしては“ENDIF”というようなものなのである．

“BEGIN”～“UNTIL”／“REPEAT”

“BEGIN”は単なるラベルである．“UNTIL”でスタックトップの真偽を検査し，真であれば“BEGIN”の位置に戻る．これは，ある条件を満たすまで実行を繰り返すようなロジックの実現に使用する．“UNTIL”ではスタックトップの真偽の検査のためにデータを1つ消費するので，ここにくるまでに真偽値を積んでおかなければならない．

“REPEAT”もまた“BEGIN”と組み合わせて使用するが，こちらは無条件で“BEGIN”の位置に戻る無限ループを構成するものである．無限ループからの強制脱出には“RETURN”というワードを使用する．これはリターンスタックトップの1ワードを捨てるワードである．

“DO”～“LOOP”

“BEGIN”～“UNTIL”は条件を満たすまでの繰り返しだが，こちらは決まった回数の繰り返しである．“DO”はスタックトップの数値と0をリターンスタックに積む．スタックトップの数値はループ回数であり，0はループカウンタである．“LOOP”でリターンスタック上のループカウンタをインクリメントし，ループ回数に達したらリターンスタック上の2つのデータを捨て，ループを打ち切る．

ループカウンタは“i”というワードで参照することができ，“j”というワードでループ回数を知ることができる．ループの途中で強制的に脱出したいときは“LEAVE”というワードを呼び出す．これはループカウンタを強制的にループ回数に一致させるワードである．ループの脱出は“LEAVE”の位置ではなく“LOOP”で行われるので注意すること．

■コンパイルとオブジェクト生成

以下の説明で「特殊なワード」という表現を用いるが，これはコンパイラが暗黙のうちにそれへの呼び出しコードを生成するワードという意味で，ワード自体が特殊というわけではない．

(1) 数値

“$$WNUMBER”とは，自分自身

リスト1 FAST.F

```
; ================================================================
;
;                    【FASTコンパイラ】
;
;        Copyright(C) 1991 By Justech Labo. H.Kanoshima
;
; ================================================================

include         core.h

_TEXT   SEGMENT WORD PUBLIC 'CODE'
_TEXT   ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS

_STACK  SEGMENT STACK
_STACK  ENDS

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT , DS:_DATA

; ==========================================================================
;
;                              CONSTANT
;
; ==========================================================================
;
TRUE            equ     1
FALSE           equ     0

STDIN           equ     0
STDOUT          equ     1
STDERR          equ     2

TAB             equ     09h

NULLWORD        equ     1
SQUOTE          equ     2
NUMBER          equ     3
DQUOTE          equ     4
STRING          equ     5
ADDRESS         equ     6
WORDNAME        equ     7
OPENWORD        equ     8       ; "["
CLOSEWORD       equ     9       ; "]"
C_BEGIN         equ     10
C_UNTIL         equ     11
C_REPEAT        equ     12
C_DO            equ     13
C_LOOP          equ     14
C_IF            equ     15
C_ELSE          equ     16
C_THEN          equ     17

IFTHEN          equ     0000h
BEGINUNTIL      equ     1000h
DOLOOP          equ     2000h
;
```


ワンラインNEWS
アシストはノート型パソコン用の個人情報管理ソフト「ゆーぴーメモ INFO SELECT」を発売．これは米国のソフトを日本語化したもので，東芝J-3100シリーズ対応．

```
; ======================================================================
;
;                                   SMALL DEFINITIONS (I)
;
; ======================================================================
;
[ sprint_crlf                           ; string | .
        sprint crlf ]

[ print_;                               ; . | .
        ';' cprint ]

[ print_$                               ; . | .
        '$' cprint ]

[ print_TAB                             ; . | .
        (TAB) cprint ]

[ print_TAB_sprint                      ; string | .
        print_TAB sprint ]

[ sprint_print_TAB                      ; string | .
        sprint print_TAB ]

[ print_TAB_sprint_print_TAB            ; string | .
        print_TAB sprint print_TAB ]

[ print_TAB_DW_TAB                      ; . | .
        "dw" print_TAB_sprint_print_TAB ]

[ print_TAB_DB_TAB                      ; . | .
        "db" print_TAB_sprint_print_TAB ]

[ print_OFFSET                          ; . | .
        "offset" sprint ]

[ print_TAB_DW_TAB_OFFSET_TAB           ; . | .
        print_TAB_DW_TAB print_OFFSET print_TAB ]

[ print_DECIMALNUMBER_h                 ; number, handle | .
        >r
        #convbuf_end 0 over c!
        >r
        begin
                dup 10 mod '0' + r> 1- dup >r c!
                10 / dup 0<>
        until
        drop
        r> r> sprint_handle
]

[ print_(DECIMALNUMBER)_h               ; number, handle | .
        >r
        '(' i cprint_handle
        i print_DECIMALNUMBER_h
        ')' r> cprint_handle
]

[ convert_hex_ascii                     ; number | char(hex)
        dup 9 >
        if      7 +
        then
        '0' +
]

[ print_HEXNUMBER                       ; number | .
        #convbuf_end 0 over c!
        4 do
                1-
                over 0fh & convert_hex_ascii
                over c!
                swap 4 >> swap
        loop
        sprint
        drop
]

[ print_$HEXNUMBER                      ; number | .
        print_$ print_HEXNUMBER
]

[ print_source_name_and_line            ; handle | .
        >r
        #source_name i sprint_handle
        #lnumber @ r> print_(DECIMALNUMBER)_h
]

[ error_print                           ; string | .
        (STDERR) print_source_name_and_line
        ' ' (STDERR) cprint_handle
        (STDERR) sprint_handle
        (STDERR) crlf_handle
]

[ ?spacer                               ; char | (TRUE)/(FALSE)
        dup ' ' = swap (TAB) = ||
]

[ skip_space                            ; address | address
        begin
                dup c@ ?spacer
                if      1+
                else    return
                then
        repeat
]

[ ?decimal_code                           char | (TRUE)/(FALSE)
        '0' '9' <range>
]

[ ?hex_code                               char | (TRUE)/(FALSE)
        dup ?decimal_code
        if      drop (TRUE) return
        else    cupcase 'A' 'F' <range>
        then
]

[ ?alnum                                  char | (TRUE)/(FALSE)
        dup ?decimal_code
        if      drop (TRUE) return
        else    cupcase 'A' 'Z' <range>
        then
]
;
; ======================================================================
;
;                                   SMALL DEFINITIONS (II)
;
; ======================================================================
;
[ c@source_ptr                          ; . | char
        #source_ptr @ c@ ]

[ c@source_ptr+                         ; . | char
        #source_ptr @1+ c@ ]

[ c@stail_word_name                     ; . | char
        #word_name c@stail ]

[ get_word_cpair                  ; char | .
        c@source_ptr+ #word_name c+
        >r
        begin
                c@source_ptr 0=
                if      "Can't find '" sprint r> cprint 27h cprint crlf
                        return
                then
                c@source_ptr+ dup #word_name c+
                i =
                if      rdrop return
                then
        repeat
]

[ get_word_sub                    ; . | .
        begin
                c@source_ptr 0=
                if      return
                then
                c@source_ptr+ dup ?spacer
                if      drop return
                then
                #word_name c+
        repeat
]

[ ?word_name                      ; string | (TRUE)/(FALSE)
        #word_nameup s= ]

[ supcase                         ; data , buffer | .
        >r
        0 i c!
        begin
                dup c@ dup 0=
                if      drop drop rdrop return
                else    cupcase i c+
                        1+
                then
        repeat
]

[ check_word_attr                 ; . | word attribute
        #word_name #word_nameup supcase
        "IF" ?word_name
        if      (C_IF)          return
        then
        "ELSE" ?word_name
        if      (C_ELSE)        return
        then
        "THEN" ?word_name
        if      (C_THEN)        return
        then
        "BEGIN" ?word_name
        if      (C_BEGIN)       return
        then
        "UNTIL" ?word_name
        if      (C_UNTIL)       return
        then
        "REPEAT" ?word_name
        if      (C_REPEAT)      return
        then
        "DO" ?word_name
        if      (C_DO)          return
        then
        "LOOP" ?word_name
        if      (C_LOOP)        return
        then
        "]" ?word_name
        if      (CLOSEWORD)     return
        then
        "[" ?word_name
        if      (OPENWORD)      return
        then

        (WORDNAME)
]

[ !word_attr                            ; attr | .
        #word_attr ! ]
```

ワンラインNEWS
東芝は先端メモリ量産工場として建設している四日市工場で16MビットDRAMに対応したウェハーラインをすべて8インチ化する．当初は6インチを予定していた．

を呼び出したコードの次にあるワード値をスタックに積む特殊なワードである．上位ワードには自動的にその時点のDSが入る（図1）．

(2) アドレス（オフセット）

図2参照のこと．

(3) 分岐ワード

分岐ワードのコンパイルには分岐構文スタックという特別なスタックを使用する．これは分岐先のラベルの生成と，分岐構文のペアの確認のためのものである．コンパイラによって生成されるラベルは0からの連続番号であり，これを十六進4桁の文字列に変換して先頭に‘$’をつけたものを実際のラベルとして使用する．

分岐構文スタックの状態図を本文といっしょに見ていただくと理解しやすいだろう．

“IF”～“ELSE”～“THEN”

【IF】

“f_$$JMPFALSE”は，スタックトップが偽（ゼロ）であれば自分を呼び出したコードの次に書かれているワード値をオフセットとしてそこに実行を移す特殊なワードである．

“IF”を見つけると新しいラベル名を用意し，“IF”を表す識別コードとともに分岐構文スタックに積む．次に“f_$$JMPFALSE”への呼び出し命令と，“ELSE”または“THEN”の位置を参照するラベルを生成する（図3）．

【ELSE】

“JMP”命令は“THEN”の位置へのものである．その直後に生成されるラベルは“IF”から参照される．なお，“ELSE”はなくてもかまわない．

“ELSE”を見つけると，まず分岐構文スタックのトップを見て，それが“IF”の識別コードをもっているかどうかを確認する．合わなければエラーである．OKならば新しいラベル名を用意して，それへのジャンプ命令を生成する．次に分岐構文スタックからラベルを取り出し，“IF”からの飛び先のラベルを生成する．最後に新しく用意したラベルを“IF”の識別コードとともに分岐構文スタックに積む（図4）．

【THEN】

オブジェクトの生成はない．このラベルは“ELSE”から，“ELSE”がなければ“IF”から参照される．

“THEN”を見つけるとまず分岐構文スタックのトップを見て，それが“IF”の識別コードをもっているかどうかを確認する．合わなければエラーである．OKならば分岐構文スタックからラベルを取り出し，“IF”あるいは“ELSE”からの飛び先を生成する（図5）．

“BEGIN”～“UNTIL”／“REPEAT”

【BEGIN】

オブジェクトの生成はない．このラベルは“UNTIL”，または“REPEAT”から参照される．

“BEGIN”を見つけると新しいラベル名を用意し，“BEGIN”を表す識別

図1　数値

```
      ソース：100
コード生成：CALL        f_$$WNUMBER
            DW          100
      ソース：(STACK_BASE + 3)
コード生成：CALL        f_$$WNUMBER
            DW          (STACK_BASE + 3)
```

図2　アドレス

```
      ソース：DATA_BUFFER
コード生成：CALL        f_$$WNUMBER
            DW          OFFSET DATA_BUFFER
```

図3　分岐ワードIF

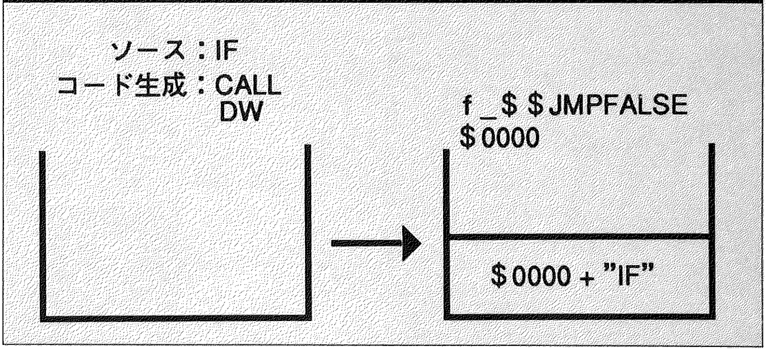
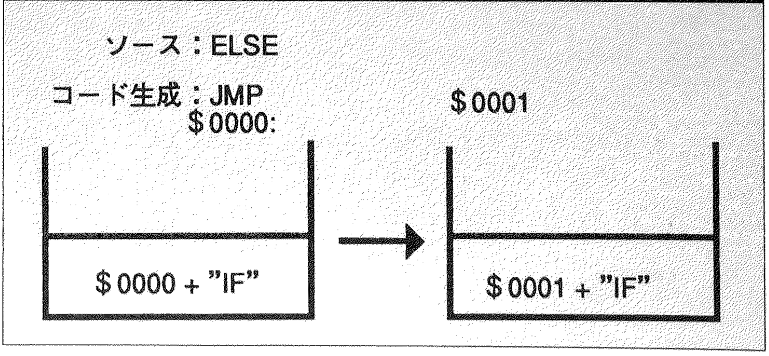


図4　分岐ワードELSE



ワンラインNEWS
通産省と郵政省共管の基盤技術研究促進センターは，有機化合物の赤外線透過特性などスペクトルデータを収めたCD-ROMを発売．25,500種の化合物データが入っている．

```

[ @word_attr                            ; . | attr
        #word_attr @ ]

[ ?word_attr                            ; attr | (TRUE)/(FALSE)
        @word_attr = ]

[ get_word_source_ptr_to_word_name      ; . | .
        #source_ptr @ skip_space #source_ptr !
        0 #word_name c!
        c@source_ptr >r
        i 0= i ';' = ||
        if      (NULLWORD)  !word_attr
                rdrop
                return
        then
        i 27h =
        if      27h get_word_cpair
                (NUMBER) !word_attr
                rdrop
                return
        then
        i '"' =
        if      '"' get_word_cpair
                (STRING) !word_attr
                rdrop
                return
        then
        i '(' =
        if      ')' get_word_cpair
                (NUMBER) !word_attr
                rdrop
                return
        then
        i '#' =
        if      get_word_sub
                (ADDRESS) !word_attr
                rdrop
                return
        then
        rdrop
        get_word_sub
        c@stail_word_name cupcase 'H' =
        if      check_hex_number_at_word_name
        else    check_decimal_number_at_word_name
        then
        if      (NUMBER)
        else    check_word_attr
        then
        !word_attr
]
;
; ======================================================================
;
;                          ASSEMBLER MODE
;
; ======================================================================
;
[ print_forth_headder                          ; . | .
        #proc_name sprint_print_TAB "proc" sprint_crlf  ; <name>  proc
        #word_name c@ '_' =
                if      return
                then
        "if"      sprint_print_TAB "0" sprint_crlf      ; if    0
        "extrn"   sprint_print_TAB
        #proc_name sprint ":near" sprint_crlf           ; extrn   <name>:near
        "endif"   sprint_crlf                           ; endif
        "public"  sprint_print_TAB
        #proc_name sprint_crlf                          ; public  <name>
]

[ check_hex_number_at_word_name         ; . | (TRUE)/(FALSE)
        #word_name
        begin
                dup c@ dup cupcase 'H' =
                if      drop drop (TRUE) return
                else    ?hex_code 0=
                        if      drop (FALSE) return
                        then
                        1+
                then
        repeat
]

[ ?label_char                           ; char | (TRUE)/(FALSE)
        dup ?alnum
        if      drop (TRUE) return
        then
        >r
        i '$' =
        i '_' = ||
        r> '?' = ||
]

[ check_decimal_number_at_word_name     ; . | (TRUE)/(FALSE)
        #word_name
        begin
                dup c@ dup 0=
                if      drop drop (TRUE) return
                else    ?decimal_code 0=
                        if      drop (FALSE) return
                        then
                        1+
                then
        repeat
]

[ convert_special_character             ; char , address | . (_HH_)
        >r
        '_' i c+
        dup 4 >> convert_hex_ascii i c+
        0fh &    convert_hex_ascii i c+
        '_' r> c+
]

[ convert_word_name_to_label_name       ; . | .
        "f_" #label_name s!
        #word_name
        begin
                dup c@ dup 0=
                if      drop drop return
                else    dup ?label_char
                        if      #label_name c+
                        else    #label_name convert_special_character
                        then
                        1+
                then
        repeat
]

[ copy_label_name_to_proc_name          ; . | .
        #label_name #proc_name s! ]

[ set_forth_mode                        ; . | .
        1 #forth_mode !
        reset_lstack ]

[ reset_forth_mode                      ; . | .
        0 #forth_mode ! ]

[ ?forth_mode                           ; . | (TRUE)/(FALSE)
        #forth_mode @ ]

[ proc_asm      ; . | .
        #source_line c@ '[' =
        if      get_word_source_ptr_to_word_name
                get_word_source_ptr_to_word_name
                convert_word_name_to_label_name
                copy_label_name_to_proc_name
                print_forth_headder
                set_forth_mode
                reset_lstack
        else
                #source_line sprint_crlf
        then
]

;
; ======================================================================
;
;                          FORTH(FAST) MODE
;
; ======================================================================
;
[ check_label           ; data , check_pattern | (TRUE)/(FALSE)
        swap 0f000h & <>
        if      "LSTACK missmatch" error_print
        then
]

[ get_newlabel          ; . | label number
        #llabel @1+
]

[ push_lstack           ; label number | .
        #lstackp @ #lstack =
        if      drop "LSTACK overflow" error_print
        else    #lstackp @ 2 - #lstackp !
                #lstackp @ !
        then
]

[ pop_lstack            ; . | label number
        #lstackp @ #lstackp =
        if      "LSTACK underflow" error_print
                (FALSE)
        else    #lstackp @ @
                #lstackp @ 2 + #lstackp !
        then
]

[ reset_lstack          ; . | .
        #lstackp #lstackp !
]

[ check_lstack          ; . | .
        #lstackp @ #lstackp <>
        if      "LSTACK unbalance" error_print
                reset_lstack
        then .
]

[ print_source_line_in_forth_mode       ; . | .
        print_;
        (STDOUT) print_source_name_and_line
        ' ' cprint
        #source_line sprint crlf
]

; =============== COMPILE WORD ========================

[ compile_closeword                     ; . | .
        "ret" print_TAB_sprint crlf
        #proc_name sprint
        "endp" print_TAB_sprint crlf
        reset_forth_mode
]

[ compile_call_word                     ; string | .
        "call" print_TAB_sprint
```

コードとともに分岐構文スタックに積む．次に，このラベル名をこの位置に生成する（図6）．

【UNTIL】

"f_$$JMPTRUE"は，スタックトップが真（ゼロ以外）であれば自分を呼び出したコードの次に書かれているワード値をオフセットとしてそこに実行を移す特殊なワードである．

"UNTIL"を見つけるとまず分岐構文スタックのトップを見て，それが"BEGIN"の識別コードをもっているかどうかを確認する．合わなければエラーである．OKならば"f_$$JMPTRUE"への呼び出し命令を生成し，次に分岐構文スタックからラベルを取り出し，"BEGIN"の位置を参照するラベル名を生成する（図7）．

【REPEAT】

無条件に"BEGIN"の位置にジャンプする命令を生成する．

"REPEAT"を見つけるとまず分岐構文スタックのトップを見て，それが"BEGIN"の識別コードをもっているかどうかを確認する．合わなければエラーである．OKならば分岐構文スタックからラベルを取り出し，"BEGIN"の位置へのジャンプ命令を生成する（図8）．

"DO"～"LOOP"

【DO】

"f_$$DO_ENTRY"は，スタックトップの数値と0の2つをリターンスタックに積む特殊なワードである．0はループカウンタとして使用され，"LOOP"の位置でインクリメントされる．この数値がループ回数に達したら，繰り返しをやめる．

"DO"を見つけると新しいラベル名を用意し，"DO"を表す識別コードとともに分岐構文スタックに積む．"f_$$DO_ENTRY"への呼び出し命令を生成し，次に"LOOP"から参照されるラベル名をこの位置に生成する（図9）．

【LOOP】

"f_$$LOOP_ENTRY"はリターン

図5　分岐ワードTHEN

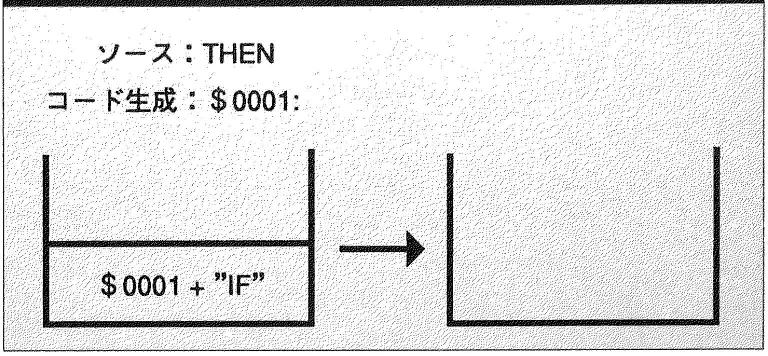


図6　分岐ワードBEGIN

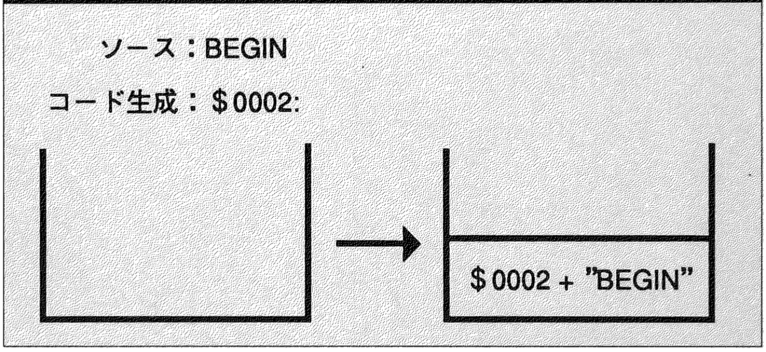


図7　分岐ワードUNTIL

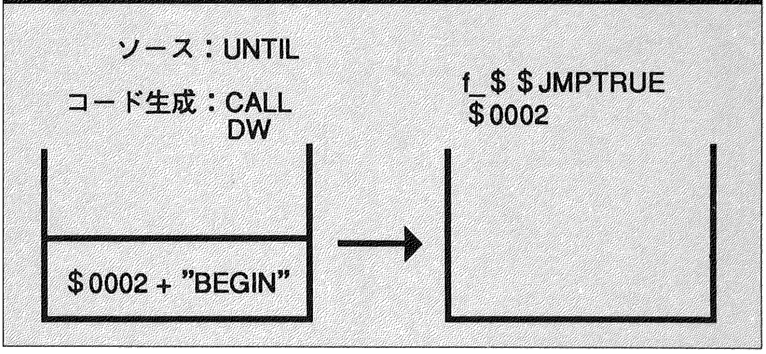
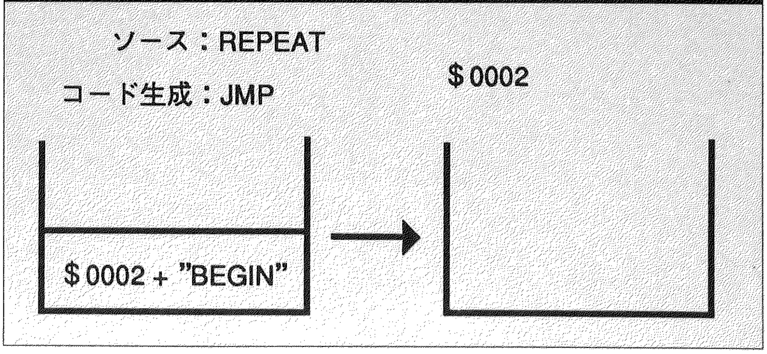


図8　分岐ワードREPEAT



ワンラインNEWS
東芝は自社のノート型パソコン「ダイナブック」シリーズをLANに接続するための専用アダプタ「Dynaイーサアダプタ」を発売した．従来は拡張アダプタが必要だった．

```
        print_TAB_sprint crlf
]

[ compile_number_word
        "f_$$wnumber" compile_call_word
]

[ compile_number                ; . | .
        compile_number_word
        print_TAB_DW_TAB
        #word_name sprint crlf
]

[ compile_var_address           ; . | .
        compile_number_word
        print_TAB_DW_TAB_OFFSET_TAB
        #word_name 1+ sprint crlf
]

[ compile_string                ; . | .
        "f_$$cstring" compile_call_word
        print_TAB_DW_TAB
        #word_name slen 1-
         (STDOUT) print_DECIMALNUMBER_h crlf
        print_TAB_DB_TAB
        #word_name slen 2 >
        if      #word_name sprint ',' cprint
        then
        '0' cprint crlf
]

[ compile_label                 ; label number | .
        print_$HEXNUMBER ':' cprint crlf
]

[ compile_if                    ; . | .
        "f_$$jmpfalse" compile_call_word
        print_TAB_DW_TAB
        get_newlabel (IFTHEN) +
        dup print_$HEXNUMBER crlf
        push_lstack
]

[ compile_else                  ; . | .
        "jmp" print_TAB_sprint_print_TAB
        get_newlabel (IFTHEN) +
        dup print_$HEXNUMBER crlf
        pop_lstack dup (IFTHEN) check_label
        compile_label
        push_lstack
]

[ compile_then                  ; . | .
        pop_lstack dup (IFTHEN) check_label
        compile_label
]

[ compile_begin                 ; . | .
        get_newlabel (BEGINUNTIL) + dup
        compile_label
        push_lstack
]

[ compile_until                 ; . | .
        "f_$$jmptrue" compile_call_word
        pop_lstack dup (BEGINUNTIL) check_label
        print_TAB_DW_TAB print_$HEXNUMBER crlf
]

[ compile_repeat                ; . | .
        "jmp" print_TAB_sprint_print_TAB
        pop_lstack dup (BEGINUNTIL) check_label
        print_$HEXNUMBER crlf
]

[ compile_do                    ; . | .
        "f_$$do_entry" compile_call_word
        get_newlabel (DOLOOP) + dup
        compile_label
        push_lstack
]

[ compile_loop                  ; . | .
        "f_$$loop_entry" compile_call_word
        pop_lstack dup (DOLOOP) check_label
        print_TAB_DW_TAB print_$HEXNUMBER crlf
]

[ compile_openword              ; . | .
        "'[' in forth mode" error_print
]

[ compile_word                  ; . | .
        @word_attr (NUMBER)   = if    compile_number          return then
        @word_attr (ADDRESS)  = if    compile_var_address     return then
        @word_attr (STRING)   = if    compile_string          return then
        @word_attr (C_IF)     = if    compile_if              return then
        @word_attr (C_ELSE)   = if    compile_else            return then
        @word_attr (C_THEN)   = if    compile_then            return then
        @word_attr (C_BEGIN)  = if    compile_begin           return then
        @word_attr (C_UNTIL)  = if    compile_until           return then
        @word_attr (C_REPEAT) = if    compile_repeat          return then
        @word_attr (C_DO)     = if    compile_do              return then
        @word_attr (C_LOOP)   = if    compile_loop            return then
        @word_attr (OPENWORD) = if    compile_openword        return then

        #label_name compile_call_word
]

[ proc_forth            ; . | .
        #source_line c@ '*' =
        if      #source_line 1+ sprint_crlf return
        then
        print_source_line_in_forth_mode
        begin
                get_word_source_ptr_to_word_name
                convert_word_name_to_label_name
                (NULLWORD) ?word_attr
                if      return
                else    (CLOSEWORD) ?word_attr
                        if      compile_closeword       return
                        else    compile_word
                        then
                then
        repeat
]

;
; ======================================================================
;
;                               MAIN
;
; ======================================================================
;
[ setup_source_name     ; address | .
        begin
                dup c@ dup 21h <
                if      drop drop return
                else    #source_name c+
                        1+
                then
        repeat
]

[ setup_sourcefile      ; . | .
        0 #source_name c!
        my_psp 81h + skip_space dup c@ 0dh =
        if      drop
                (STDIN) #source_fp !
        else    setup_source_name
                #source_name 0 fopen
                if      #source_fp !
                else    drop
                        #source_name sprint " - Can't open" sprint_crlf
                        1 exit
                then
        then
]

[ read_source_line      ; . | (TRUE)/(FALSE)
        #lnumber 1+!                            ; increment line number
        #source_line #source_ptr !              ; init source line pointer
        #source_line #source_fp @ fsinput       ; read new source line
]

[ system_initialize     ; . | .
        reset_forth_mode
        reset_lstack
        0 #llabel !
]

[ main
        $$startup
        system_initialize
        setup_sourcefile
        begin
                ?pstack_initial 0=
                if      "Stack error" sprint cinput
                then
                read_source_line
                if      ?forth_mode
                        if      proc_forth
                        else    proc_asm
                        then
                else    return
                then
        repeat
]

_TEXT   ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'

source_fp       dw      (?)
source_name     db      80 dup (?)
source_line     db      256 dup (?)
source_ptr      dw      (?)

word_attr       dw      (?)
word_name       db      128 dup (?)
word_nameup     db      128 dup (?)
label_name      db      128 dup (?)
proc_name       db      128 dup (?)

forth_mode      dw      (?)

lnumber         dw      (?)

convbuf         db      16 dup (?)
convbuf_end     db      (?)

lstack          dw      16 dup (?)
lstackp         dw      (?)

llabel          dw      (?)

_DATA   ENDS

        END     f_main
```

ワンラインNEWS
日本電気のマイクロエレクトロニクス研究所は256MビットDRAM実現のための微細配線技術を開発。93〜94年をメドにチップ化したいとしている。

スタック上の2つの数値，すなわちループ回数とループカウンタを比較して，ループを続けるかどうかを判断する．ループカウンタはこのワードがインクリメントする．

“LOOP”を見つけるとまず分岐構文スタックのトップを見て，それが“DO”の識別コードをもっているかどうかを確認する．合わなければエラーである．OKならば“f_$$LOOP_ENTRY”への呼び出し命令を生成し，次に分岐構文スタックからラベルを取り出し，“DO”の位置を参照するラベル名を生成する（図10）．

(4) FAST記述の開始と終了

ラベル宣言のみである（図11）．“if”～“endif”の間の“extrn”宣言は，条件式が偽になっているので，このソースのコンパイル中には無視される．これはあとで“FIND.EXE”というMS-DOSに付属しているツールを使って“entrn”宣言の行だけを取り出し，このソースファイルのヘッダファイルを作成するためである．

なお，先頭がアンダースコアで始まるワードはそのソースファイル内だけで参照されるローカルなワードと解釈するので，“entrn”ならびに“public”宣言は生成されない．

(5) 一般ワード

ワード（一般の言語の関数に相当するもの）の呼び出しはCPUのCALL命令によって行う（図12）．

以上がFASTのオブジェクト生成の概要である．不明な点は実際にプログラムを書いてFASTに通してみればわかると思う．また，コンパイラ自体のソースも公開しているので，そちらを読んでみてもいいだろう．コンパイラといってもFASTは非常に単純な構成なので，大まかな処理の流れをつかむのにさほど時間はかからないと思う．

■コンパイルのしかた

FASTが自分自身をコンパイルしてできたMASMのソースファイルがあるので，これをアセンブルすればFASTコンパイラの.OBJファイルができる．これに来月号で掲載する基本ライブラリ“CORE.OBJ”をリンクすれば，FASTコンパイラが完成するわけである．

編集部注：紙面の都合でMASMのソースファイルは掲載しておりません．付録のディスクには収録されておりますので，そちらをご利用ください．

参考文献：

FORTH入門（レオ・ブロディー著，原道宏訳，工学社）
標準FORTH(井上外志雄著，共立出版)

図9　分岐ワードDO

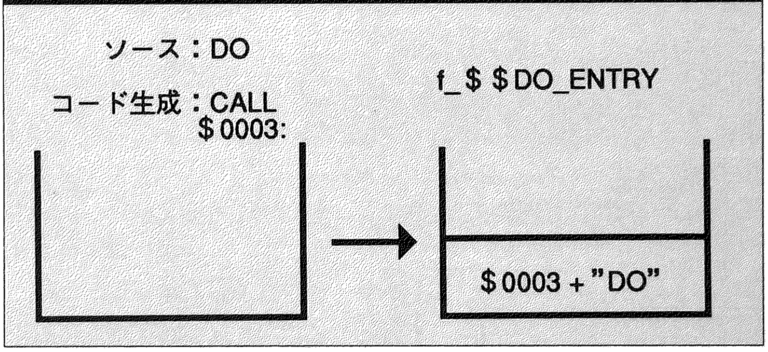


図10　分岐ワードLOOP

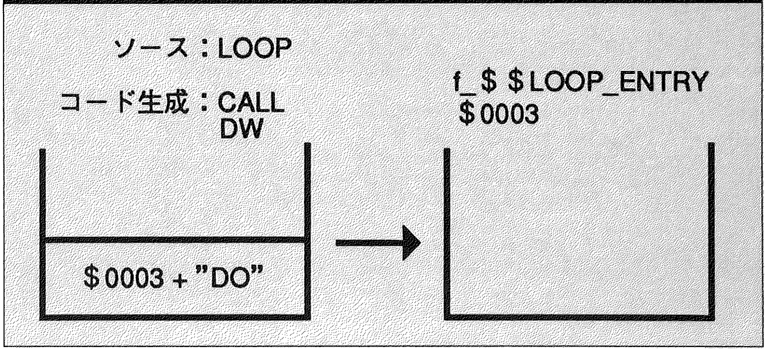


図11　開始と終了

```
        ソース：[ PROC_NAME

  コード生成：f_PROC_NAME          proc
              if                 0
              extrn              f_print_$:near
              endif
              public             f_print_$

        ソース：]

  コード生成：RET
              f_PROC_NAME          endp

     "RET"命令と"endp"を生成する。
```

図12　一般ワード

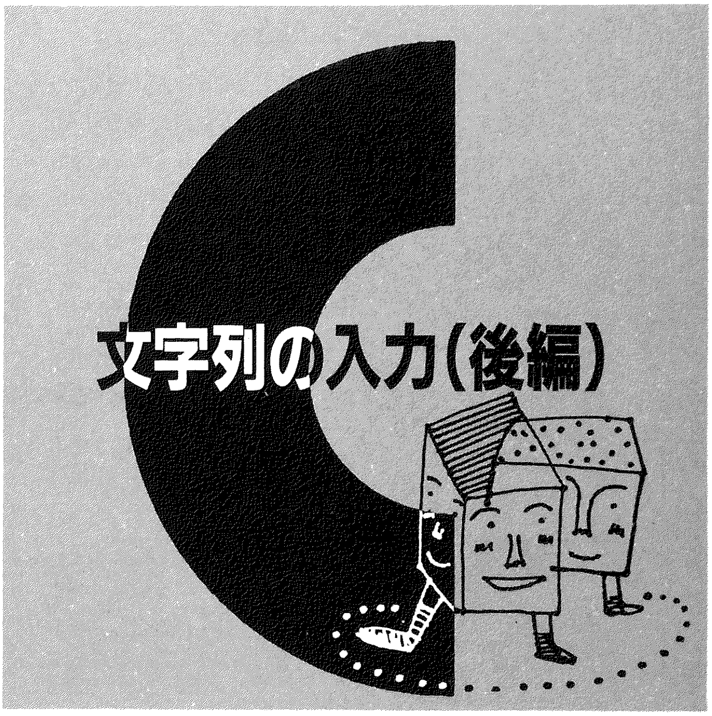
```
        ソース：ABCD

  コード生成：CALL          f_ABCD
```


ワンラインNEWS
リコーは同社の統合型ソフト「マイツール」を日本IBMのOS「DOS/V」向けに移植する．同社は日本IBMからOEM調達により「PS/55」を販売している．

Illustration:
Masami Nozue

実践C塾

基礎テクニック篇　第3回



二代目塾頭　大野元久

今回は，前回作成した文字列入力ルーチンにヒストリー機能を追加し，さらに，入力文字種を制限したりFEP（かな漢字変換フロントエンドプロセッサ)を制御する処理を追加する．また，これらの文字列入力ルーチン，FEP制御ルーチンを利用して基本的な編集機能を備えた簡易エディタを作成する．

ソフトウェアを利用するときに，同じ文字列を何度も入力しなければならないというのはわずらわしいものです．コンピュータというものは記憶することが得意なのですから，こうしたことを記憶しておき，ユーザーの手間を減らすというのはよいアプリケーションの条件ともいえるでしょう．ここで追加するヒストリー機能とは，こうした機能を実現するものです．

前回作成したeditstr.cを改良してヒストリー機能を組み込んだものがeshist.c(リスト1)，editstr.h(リスト2)です．editstr.hは，ヒストリー機能と後述するFEP制御や入力文字種の制限のために列挙型editstr_tと列挙型edittype_tについてシンボル定義が変更されていますので注意してください．前回同様，eshist.cを使うためにはkbd.c，tvram.cなどが必要になりますが，今回は付録ディスクのみでこれらを提供し，リストは掲載しません．

eshist.cの使用例をehtest.c（リスト3）に示します．このプログラムはヒストリーバッファを指定している以外，前回示したestest.cとほとんど変わらないように見えますが，リターンキーをeditstrに渡しています．ヒストリーの処理は，editstrの中で行われていますからヒストリー文字列を選択するための上下のカーソルキー（KBD_UP，KBD_DOWN)はもちろん，ヒストリーバッファに文字列を登録するためのリターンキー（KBD_CR）をeditstrに渡さなければヒストリーは正しく機能しません．

リスト1　(eshist.c)

```
/*
 *  eshist.c: 文字列入力モジュール（ヒストリー機能つき）
 *      written by Motohisa Ohno
 */
#include <string.h>
#include <jctype.h>
#include <jstring.h>
#include <dos.h>
#include "turboc.h"
#include "kbd.h"
#include "tvram.h"
#include "editstr.h"

int insert_mode = 1;                /* 挿入モード */
unsigned edit_attr = 0x00E1;        /* 入力文字の属性 */

/*
 *  文字列入力ルーチン
 *      入力    code: 入力された文字コードあるいはEDS_BEGIN
 *              ep: 文字列入力構造体（EDITSTR）へのポインタ
 *      出力    文字列入力の状態
 */
unsigned editstr(unsigned code, EDITSTR *ep)
{
    char *p;
    int len;
    int i, n, m;

    len = strlen(ep->bufp);
    switch (code) {
    case EDS_BEGIN:
        /* 文字入力を行なうための初期化処理 */
        ep->histn = 0;
        /* no break */

    case EDS_REINIT:
        /* ヒストリー機能のための再初期化 */
        if (len < ep->width) {
            ep->offset = 0;
        } else {
            ep->offset = len - ep->width + 1;
        }
        ep->curp = len;
        break;

    case KBD_RIGHT:
        /* カーソルを右へ移動する */
        if (ep->curp < len) {
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            ep->curp += n;
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            if (ep->width < (ep->curp - ep->offset + n))
                ep->offset = ep->curp - ep->width + n;
        }
        break;

    case KBD_BS:
        /* バックスペース  =  左へ移動+文字の削除 */

    case KBD_LEFT:
        /* カーソルを左へ移動する */
        if (ep->curp > 0) {
            ep->curp--;
            if (nthctype(ep->bufp, ep->curp) == CT_KJ2)
                ep->curp--;
            if (ep->offset > ep->curp)
                ep->offset = ep->curp;
        }
        if (code == KBD_LEFT) break;

    case KBD_DEL:
        /* カーソル位置の文字を削除 */
        if (ep->curp < len) {
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            for (i = ep->curp + n; i < len; i++)
                ep->bufp[i-n] = ep->bufp[i];
            ep->bufp[len-n] = 0;
        }
        break;

    case KBD_INS:
        /* 挿入モードの切り換え */
        insert_mode = !insert_mode;
        break;

    case KBD_HOME:
        /* 文字列の先頭へ移動 */
        ep->curp = 0;
        ep->offset = 0;
        break;

    case KBD_CLR:
        /* 文字列の最後へ移動 */
        if (len < ep->width) {
            ep->offset = 0;
        } else {
            ep->offset = len - ep->width + 1;
        }
        ep->curp = len;
        break;

    case KBD_DOWN:
        /* ヒストリー番号を小さくする */
        if (ep->histn) --ep->histn;
        /* no break */
    case KBD_UP:
        if (ep->histp == NULL) break;

        /* ヒストリー番号を大きくする */
        if (code == KBD_UP) ++ep->histn;
        if (ep->histn) {
            /* ヒストリーからｈｎ番目の文字列を呼び出す */
            for (i = 0, p = ep->histp; i < (ep->histn - 1); i++) {
                while (*p && *p != '¥xff')
                    ++p;
                if (*p) break;
                ++p;
            }
            if (*p == '¥xff') {
                if (code == KBD_UP) --ep->histn;
                break;
```


ワンラインNEWS
日本コンピュータ工業は独自の通信ソフトを内蔵し，プリンタを直接LANに接続するプリンタボックス「JC-CONNECT MODEL101」を発売した．

```
            }
            strncpy(ep->bufp, p, ep->length);
        } else {
            /* 0番目のときは空文字列を設定 */
            ep->bufp[0] = 0;
        }
        /* 文字列入力を再初期化 */
        editstr(EDS_REINIT, ep);
        break;

    case KBD_CR:
        /* 入力文字列が空のときはヒストリーに登録しない */
        if (ep->bufp[0] != 0 && ep->histp != NULL) {
            /* ヒストリー文字列の登録 */
            /* 入力文字列と同じ文字列があればヒストリーから削除 */
            for (p = ep->histp; *p != '¥xff'; p += n) {
                n = strlen(p) + 1;
                if (strcmp(p, ep->bufp) == 0) {
                    while (*(p + n) != '¥xff') {
                        *p = *(p + n);
                        ++p;
                    }
                    *p = '¥xff';
                    break;
                }
            }
            /* 入力文字列をヒストリーの先頭に格納 */
            n = strlen(ep->bufp) + 1;
            for (i = ep->histlen - 1 - n; i >= 0; i--)
                ep->histp[i+n] = ep->histp[i];
            strcpy(ep->histp, ep->bufp);
            for (i = n; i < ep->histlen - 1; i++) {
                if (ep->histp[i] == '¥xff') break;
                if (ep->histp[i] == 0) n = i + 1;
            }
            if (i == (ep->histlen - 1)) ep->histp[n] = '¥xff';
        }
        break;

    case KBD_NULL:
    case KBD_ESC:
        /* 入力文字列としては使わない */
        break;

    default:
        /* リターン、ESC以外の通常文字は、すべて入力の対象となる */
        if (code == 0 || (KBD_BIAS <= code && code < 0x8000))
            break;
        /* 初期化直後に文字が入力されたときは */
        /* デフォルト文字列をクリアしてから入力する */
        if (ep->first) {
            ep->curp = 0;
            ep->offset = 0;
            ep->bufp[0] = 0;
            len = 0;
        }
        n = (code < 0x8000) ? 1 : 2;
        m = (insert_mode ? 0 : (iskanji(ep->bufp[ep->curp]) ? 2 : 1));
        if ((len + n - m) >= ep->length)
            break;
        if (m < n) {
            for (i = len; i >= ep->curp; i--)
                ep->bufp[i+n-m] = ep->bufp[i];
        } else if (m > n) {
            for (i = ep->curp-n+m; i <= len; i++)
                ep->bufp[i+n-m] = ep->bufp[i];
        }
        if (n > 1) ep->bufp[ep->curp++] = (code >> 8);
        ep->bufp[ep->curp++] = (unsigned char)code;
        if (ep->width <= (ep->curp - ep->offset))
            ep->offset = ep->curp - ep->width + 1;
        break;
    }
    ep->first = (code == EDS_BEGIN);
    if (code != KBD_UP && code != KBD_DOWN && code != EDS_REINIT)
        ep->histn = 0;
    /* 編集文字列の表示 */
    if (nthctype(ep->bufp, ep->offset) == CT_KJ2) {
        dspchr(ep->lx, ep->ly, ' ', edit_attr);
        dspstr(ep->lx+1, ep->ly, ep->width-1, ep->bufp+ep->offset+1,edit_attr);
    } else {
        dspstr(ep->lx, ep->ly, ep->width, ep->bufp + ep->offset, edit_attr);
    }
    /* カーソル位置の設定 */
    setcurpos(ep->lx + ep->curp - ep->offset, ep->ly);
    /* 編集状態の設定 */
    len = strlen(ep->bufp);
    n = 0;
    if (ep->offset) n |= EDS_LEFT;
    if ((ep->offset + ep->width) < len) n |= EDS_RIGHT;
    return n;
}
```

リスト2 (editstr.h)

```
/*
 * editstr.h: 文字列入力モジュールのための定義
 *     written by Motohisa Ohno
 */

typedef struct {
    int lx, ly;      /* 編集位置の開始座標：左上が（0, 0） */
    int width;       /* 編集位置の幅 */
    int length;      /* 文字バッファの最大長（ヌル文字を含む） */
    int offset;      /* 表示位置オフセット（先頭＝0） */
    int curp;        /* 現在のカーソル位置（先頭＝0） */
    int first;       /* 編集の最初かどうかのフラグ（最初＝1） */
    int type;        /* 編集文字列の種類 */
    char *bufp;      /* 文字列バッファ */
    int histn;       /* ヒストリー番号 */
    int histlen;     /* ヒストリーバッファの長さ */
    char *histp;     /* ヒストリーバッファへのポインタ */
} EDITSTR;

/* 編集状態 */
enum editstr_t {
    /* 文字列編集の最初 */
    EDS_BEGIN = 0x0800,
    /* ヒストリーのための初期化 */
    EDS_REINIT = 0x0801,
    /* 文字列編集の終り */
    EDS_END = 0x0802,

    /* 編集領域の左に文字がある */
    EDS_LEFT = 0x1000,
    /* 編集領域の右に文字がある */
    EDS_RIGHT = 0x2000,
    /* 挿入モードがオン */
    EDS_INSERT = 0x4000,
};

enum edittype_t {
    /* 編集する文字の種類 */
    EDT_DIGIT = 0x0001,
    EDT_ALPHA = 0x0002,
    EDT_KANA = 0x0004,
    EDT_SPACE = 0x0008,
    EDT_OTHER = 0x0010,
    EDT_HANKAKU = 0x001F,

    EDT_K_DIGIT = 0x0020,
    EDT_K_ALPHA = 0x0040,
    EDT_K_KATA = 0x0080,
    EDT_K_HIRA = 0x0100,
    EDT_K_KIGOU = 0x0200,
    EDT_K_SPACE = 0x0400,
    EDT_K_OTHER = 0x0800,
    EDT_KANJI = 0x0FE0,
    EDT_ALL = 0x0FFF,

    /* FEPの制御 */
    EDT_USEFEP = 0x1000,
    EDT_NOUSEFEP = 0x2000,
};

extern int insert_mode; /* 挿入モード */

unsigned editstr(unsigned code, EDITSTR *ep);   /* 文字列入力 */
```

ヒストリープログラムの解説

editstrを拡張して，ヒストリー機能を実現することはそれほど難しくありません．ヒストリーバッファへ文字列を登録する機能と，ヒストリーバッファから文字列を取り出す機能をswitch文の中に追加すればよいだけです．

editstrは，前述したヒストリー機能のために文字コードKBD_UP，KBD_DOWN，KBD_CRおよびヒストリー用の再初期化ルーチン(KBD_REINIT)のための処理が拡張されています．ヒストリーバッファにはいくつかの文字列が入っているわけですから，何番目の文字列を取り出すかを示すために構造体EDITSTRのメンバhistnが使われます．このメンバhistnは，上下のカーソルキー以外のキーが入力されたときに初期化されます．いくつか前の文字列を選択するためには，上下のカーソルキーを連続して操作します．

ここでは，前回使用しなかったメンバhistlen，histpを使っています(C++であれば継承を使ってこのような機能の追加を自然に行うことができるのですが，“C”塾ではこの誘惑を無視します)．histlen，histpは，それぞれヒストリーバッファの長さとバッファへのポインタを指しています．ヒストリー機能を使わないときは，histpをNULLに設定します．

ヒストリーバッファは，連続したASCIZ文字列（文字コード0で終了する文字列）で構成されており，ヒストリーバッファの最後には文字コードFFHが入っています．したがって，最初に入力関数getstrを呼び出すときは，あらかじめヒストリーバッファの先頭にFFHを入れておかなければなりません．ヒストリーバッファはgethistの中で自動的に更新されます．この例ではデフォルトの文字列として空文字列を与えていますが，ヒストリーバッファにあらかじめいくつか文字列を指定しておけば，それをデフォルトの文字列として使うこともできます．

上下のカーソルキーが押されるとヒストリーバッファから文字列を取り出します．まず，ヒストリーバッファの先頭からhistn-1が示すだけの文字列をスキップします．文字列の先頭が文字コードFFHであれば，それ以上文字列はないのでスキップ処理を中断します．文字列をみつけたら，それを入力文字列バッファにコピーして自分自身を再初期化します．初期化するときにメンバhistnまで0に設定されないよう，本来の初期化処理とは別の識別子EDS_REINITを使います．

リターンキーが入力されたときは，ヒストリーバッファに文字列を登録します．まず，ヒストリーバッファの中に入力文字列バッファの内容と同じものがあればそれを削除します．次に，入力文字列の長さだけヒストリーバッファの内容を移動し，文字列を設定し，ヒストリーバッファの最後を示す文字コードFFHを設定しなおします．こう

リスト3 (ehtest.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "kbd.h"
#include "tvram.h"
#include "editstr.h"

unsigned edit_mark_attr = 0x00A1;   /* ←→マークのための文字属性 */

int getstr(EDITSTR *ep)
{
    char *p;
    unsigned code, state;

    /* ヒープメモリから編集用の一時バッファを確保 */
    p = strdup(ep->bufp);
    if (p == NULL) return 0;
    strcpy(p, ep->bufp);

    /* 文字列入力の初期化 */
    state = editstr(EDS_BEGIN, ep);
    do {
        dspchr(ep->lx-1, ep->ly,
                (state & EDS_LEFT) ? 0x1D : ' ', edit_mark_attr);
        dspchr(ep->lx+ep->width, ep->ly,
                (state & EDS_RIGHT) ? 0x1C : ' ', edit_mark_attr);
        code = kbdin();
        switch (code) {
        case KBD_ESC:
            /* ＥＳＣのときは文字列を元に戻す */
            strcpy(ep->bufp, p);
            break;
        case KBD_CR:
            /* リターンが入力されたら文字列設定 */
            /* no break */
        default:
            state = editstr(code, ep);
            break;
        }
    } while (code != KBD_CR && code != KBD_ESC);
    free(p);
    return code;
}

void main()
{
    char buf[30];
    char histbuf[200];
    EDITSTR es;
    unsigned code;

    tvraminit();
    kbdinit();

    printf("¥033[2J");  /* 画面の消去 */
    es.lx = 10;
    es.ly = 5;
    es.width = 20;
    es.length = 30;
    es.bufp = buf;
    es.histlen = 200;
    es.histp = histbuf;
    histbuf[0] = '¥xff';
    do {
        es.bufp[0] = 0;
        code = getstr(&es);
        printf("your input is [%s]¥n", es.bufp);
    } while (code != KBD_ESC);

    kbdterm();
    tvramterm();
}
```


ワンラインNEWS
東芝は7月出荷分から4MビットDRAMをすべて縮小版にする．4Mでは日立製作所が先行，東芝・日本電気を合わせた3社が激しいシェア競争を展開している．

することで，単純ですがヒストリー文字列の学習機能をサポートできます．

改良する場合はKBD_CR，KBD_UP, KBD_DOWNの処理を変更してください．ほかの機能を追加する場合も，たいていはeditstrのcase文を追加するだけで実現できるでしょう．

余談ですが，文字コードFFHとの比較をするときに0xFFとか-1という書き方をしてはいけません．代入であれば両方とも同じように処理されますが，比較の場合はchar型を符号つきか符号なしのどちらでコンパイルするかによって結果が異なるからです．'¥xFF'と書いておけば，つねに正しい文字コードの比較が行われます．(char) (-1) のように書いてもかまいませんが，ちょっとまわりくどいですね．

前述したように，ヒストリー機能を使うためにはkbdinとeditstrのループの中でKBD_CR, KBD_UP, KBD_DOWNを必ずeditstrに渡さなければなりません．たとえば，表計算ソフトのように入力位置が格子状に並んでいる場合は，各項目を移動するためにカーソルキーを使うことになるでしょうが，KBD_UP，KBD_DOWNをこのために処理してしまいeditstrに渡さなければヒストリー機能は使えません．どうしてもヒストリー機能を使いたければ，ほかのキーコードをKBD_UP，KBD_DOWNに変換してからeditstrを呼び出します．また，ファンクションキーやカーソルキーを入力したときに文字列入力を確定したりキャンセルする場合は，いったんKBD_CRかKBD_ESCをeditstrに渡さなければなりません．たとえば文字列の入力を確定してからファンクションキーの処理へ移行する場合は，editstr(KBD_CR，ep);などとします．

日本語を扱うアプリケーションでは，文字を入力する際にかな漢字変換システム（FEP）を制御しなければならないことがあります．FEPには数多くの種類があり，これらに対応することはかなりの困難がともなうものなのですが，幸いなことにほとんどのFEPをカバーする制御モジュール（FEPCTRL）が太田純氏によってフリーソフトウェアとして開発されています．

今回，作者のご厚意によりFEPCTRLを付録ディスクで提供させていただくことになりました．FEPCTRLがサポートする関数を，表1に示します．このFEPCTRLを利用してFEPの制御や入力文字列の種類を指定できるような改良を行います（リスト4）．

edstr. cは，eshist. cに改良を施したものです．まず，FEPCTRLの制御をeditstrの初期化部分とリターンキーかESCが押されたときの処理部分に追加します．ここで，FEPを制御しますので，文字列の編集を終了するときは必ずリターンキーかESCの文字コード（KBD_CR，KBD_ESC）をeditstrに渡さなければなりません．ヒストリー機能と同様にファンクションキーやカーソルキーなどで入力を確定したりキャンセルする場合は，KBD_CRかKBD_ESCをeditstrに渡さなければなりません．

edstr. cの使用例は，前出のehtest. cで構造体EDITSTR型の変数esの内容を設定している行の最後（es. histp = histbuf;の後ろ）に，リスト5の1行を追加してください．

また，main()関数の先頭と最後で，それぞれfep_init () ;とfep_term () ;を呼び出してください．変更したehtest. cは，edstr. cのほかにkbd. c,

リスト4 (edstr.c)

```
/*
 *  edstr.c: 文字列入力モジュール
 *       written by Motohisa Ohno
 */
#include <string.h>
#include <ctype.h>
#include <jctype.h>
#include <jstring.h>
#include <dos.h>
#include "turboc.h"
#include "kbd.h"
#include "tvram.h"
#include "editstr.h"

int insert_mode = 1;                /* 挿入モード */
unsigned edit_attr = 0x00E1;        /* 入力文字の属性 */

/*
 *  文字列入力ルーチン
 *      入力    code: 入力された文字コードあるいはEDS_BEGIN
 *              ep: 文字列入力構造体 (EDITSTR) へのポインタ
 *      出力    文字列入力の状態
 */
unsigned editstr(unsigned code, EDITSTR *ep)
{
	char *p;
	int len;
	int i, n, m;

	len = strlen(ep->bufp);
	switch (code) {
	case EDS_BEGIN:
		/* 文字入力を行なうための初期化処理 */
		ep->histn = 0;
		if (ep->type & EDT_USEFEP)
			fep_force_on();
		else if (ep->type & EDT_NOUSEFEP)
			fep_off();
		/* no break */

	case EDS_REINIT:
		/* ヒストリー機能のための再初期化 */
		if (len < ep->width) {
			ep->offset = 0;
		} else {
			ep->offset = len - ep->width + 1;
		}
		ep->curp = len;
```

```
        break;

    case KBD_RIGHT:
        /* カーソルを右へ移動する */
        if (ep->curp < len) {
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            ep->curp += n;
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            if (ep->width < (ep->curp - ep->offset + n))
                ep->offset = ep->curp - ep->width + n;
        }
        break;

    case KBD_BS:
        /* バックスペース  =  左へ移動+文字の削除 */

    case KBD_LEFT:
        /* カーソルを左へ移動する */
        if (ep->curp > 0) {
            ep->curp--;
            if (nthctype(ep->bufp, ep->curp) == CT_KJ2)
                ep->curp--;
            if (ep->offset > ep->curp)
                ep->offset = ep->curp;
        }
        if (code == KBD_LEFT) break;

    case KBD_DEL:
        /* カーソル位置の文字を削除 */
        if (ep->curp < len) {
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            for (i = ep->curp + n; i < len; i++)
                ep->bufp[i-n] = ep->bufp[i];
            ep->bufp[len-n] = 0;
        }
        break;

    case KBD_INS:
        /* 挿入モードの切り換え */
        insert_mode = !insert_mode;
        break;

    case KBD_HOME:
        /* 文字列の先頭へ移動 */
        ep->curp = 0;
        ep->offset = 0;
        break;

    case KBD_CLR:
        /* 文字列の最後へ移動 */
        if (len < ep->width) {
            ep->offset = 0;
        } else {
            ep->offset = len - ep->width + 1;
        }
        ep->curp = len;
        break;

    case KBD_DOWN:
        /* ヒストリー番号を小さくする */
        if (ep->histn) --ep->histn;
        /* no break */
    case KBD_UP:
        if (ep->histp == NULL) break;

        /* ヒストリー番号を大きくする */
        if (code == KBD_UP) ++ep->histn;
        if (ep->histn) {
            /* ヒストリーからｈｎ番目の文字列を呼び出す */
            for (i = 0, p = ep->histp; i < (ep->histn - 1); i++) {
                while (*p && *p != '¥xff')
                    ++p;
                if (*p) break;
                ++p;
            }
            if (*p == '¥xff') {
                if (code == KBD_UP) --ep->histn;
                break;
            }
            strncpy(ep->bufp, p, ep->length);
        } else {
            /* ０番目のときは空文字列を設定 */
            ep->bufp[0] = 0;
        }
        /* 文字列入力を再初期化 */
        editstr(EDS_REINIT, ep);
        break;

    case KBD_CR:
        /* 入力文字列が空のときはヒストリーに登録しない */
        if (ep->bufp[0] != 0 && ep->histp != NULL) {
            /* ヒストリー文字列の登録 */
            /* 入力文字列と同じ文字列があればヒストリーから削除 */
            for (p = ep->histp; *p != '¥xff'; p += n) {
                n = strlen(p) + 1;
                if (strcmp(p, ep->bufp) == 0) {
                    while (*(p + n) != '¥xff') {
                        *p = *(p + n);
                        ++p;
                    }
                    *p = '¥xff';
                    break;
                }
            }
            /* 入力文字列をヒストリーの先頭に格納 */
            n = strlen(ep->bufp) + 1;
            for (i = ep->histlen - 1 - n; i >= 0; i--)
                ep->histp[i+n] = ep->histp[i];
            strcpy(ep->histp, ep->bufp);
            for (i = n; i < ep->histlen - 1; i++) {
                if (ep->histp[i] == '¥xff') break;
                if (ep->histp[i] == 0) n = i + 1;
            }
            if (i == (ep->histlen - 1)) ep->histp[n] = '¥xff';
        }

        /* no break */
    case KBD_ESC:
    case EDS_END:
        /* FEP 制御の終了 */
        if (ep->type & EDT_USEFEP)
            fep_force_off();
        else if (ep->type & EDT_NOUSEFEP)
            fep_on();
        break;

    case KBD_NULL:
        /* 入力文字列としては使わない */
        break;

    default:
        /* リターン、ＥＳＣ以外の通常文字は、すべて入力の対象となる */
        if (code == 0 || (KBD_BIAS <= code && code < 0x8000))
            break;
        if (ep->type != EDT_ALL) {
            if (code < 0x100) {
                if (isdigit(code)) {
                    if (!(ep->type & EDT_DIGIT)) break;
                } else if (isalpha(code)) {
                    if (!(ep->type & EDT_ALPHA)) break;
                } else if (iskana(code)) {
                    if (!(ep->type & EDT_KANA)) break;
                } else if (code == ' ') {
                    if (!(ep->type & EDT_SPACE)) break;
                } else {
                    if (!(ep->type & EDT_OTHER)) break;
                }
            } else {
                if (jisdigit(code)) {
                    if (!(ep->type & EDT_K_DIGIT)) break;
                } else if (jisalpha(code)) {
                    if (!(ep->type & EDT_K_ALPHA)) break;
                } else if (jiskata(code)) {
                    if (!(ep->type & EDT_K_KATA)) break;
                } else if (jishira(code)) {
                    if (!(ep->type & EDT_K_HIRA)) break;
                } else if (jiskigou(code)) {
                    if (!(ep->type & EDT_K_KIGOU)) break;
                } else if (jisspace(code)) {
                    if (!(ep->type & EDT_K_SPACE)) break;
                } else {
                    if (!(ep->type & EDT_K_OTHER)) break;
                }
            }
        }
        /* 初期化直後に文字が入力されたときは */
        /* デフォルト文字列をクリアしてから入力する */
        if (ep->first) {
            ep->curp = 0;
            ep->offset = 0;
            ep->bufp[0] = 0;
            len = 0;
        }
        n = (code < 0x8000) ? 1 : 2;
        m = (insert_mode ? 0 : (iskanji(ep->bufp[ep->curp]) ? 2 : 1));
        if ((len + n - m) >= ep->length)
            break;
        if (m < n) {
            for (i = len; i >= ep->curp; i--)
                ep->bufp[i+n-m] = ep->bufp[i];
        } else if (m > n) {
            for (i = ep->curp-n+m; i <= len; i++)
                ep->bufp[i+n-m] = ep->bufp[i];
        }
        if (n > 1) ep->bufp[ep->curp++] = (code >> 8);
        ep->bufp[ep->curp++] = (unsigned char)code;
        if (ep->width <= (ep->curp - ep->offset))
            ep->offset = ep->curp - ep->width + 1;
        break;
    }
    ep->first = (code == EDS_BEGIN);
    if (code != KBD_UP && code != KBD_DOWN && code != EDS_REINIT)
        ep->histn = 0;
    /* 編集文字列の表示 */
    if (nthctype(ep->bufp, ep->offset) == CT_KJ2) {
        dspchr(ep->lx, ep->ly, ' ', edit_attr);
        dspstr(ep->lx+1, ep->ly, ep->width-1, ep->bufp+ep->offset+1,edit_attr);
    } else {
        dspstr(ep->lx, ep->ly, ep->width, ep->bufp + ep->offset, edit_attr);
    }
    /* カーソル位置の設定 */
    setcurpos(ep->lx + ep->curp - ep->offset, ep->ly);
    /* 編集状態の設定 */
    len = strlen(ep->bufp);
    n = 0;
    if (ep->offset) n |= EDS_LEFT;
    if ((ep->offset + ep->width) < len) n |= EDS_RIGHT;
    return n;
}
```


ワンラインNEWS
富士通はCD-ROM駆動装置を「OASYS」の最上位機種に標準装備する。ワープロの画面上で「広辞苑」などのCD-ROMソフトを参照でき，複写も可能.

tvram.c, fepctrl.cとともにコンパイル・リンクしなければなりません.

edstrを実行すると，FEPがオンになります．もちろん，FEPCTRLがサポートしているいずれかのFEPが組み込まれていなければなりません．ここで漢字を入力したり，半角カナ文字を入力すれば入力文字列として扱われます．しかし，半角英数字を入力しても入力として受けつけられていないことがわかるでしょう．同様の方法で，エディタで行番号を入力させるときに数字だけを入力したり，ファイル名の入力で半角文字だけを入力させることができます．

EDS_BEGINおよびKBD_CR, KBD_ESCにFEPの制御ルーチンを追加します．構造体EDITSTRのメンバtypeでFEPの制御が指定されている場合にかぎり，強制的にFEPをオフにするかFEPをオンにするか何もしないかが選択されます．FEPCTRLを使っているため，この関数を利用するアプリケーションの先頭と最後で，それぞれfep_init(); とfep_term(); を呼び出さなければなりません．

通常文字が入力されたときは，入力された文字コードが構造体EDITSTRのメンバtypeで指定された種類に含まれるかどうかを調べ，適当でない文字コードであれば何もせずに終了します．入力を制限するためにメンバ

表1　FEPCTRLがサポートする関数

fep_init

機能	FEPCTRLの初期化
形式	int fep_init (void) ;
説明	FEPCTRLライブラリの初期化を行います．FEPCTRLを使うアプリケーションは，最初にこの関数を呼び出さなければなりません．

fep_term

機能	FEPCTRLの終了
形式	void fep_term (void) ;
説明	FEPCTRLライブラリの使用を終了したら，アプリケーション終了前にこの関数を呼び出さなければなりません．

fep_off，fep_on

機能	FEPの一時的オフ
形式	void fep_off (void) ; void fep_on (void) ;
説明	FEPの変換状態を一時的にオフにして英数字などを入力させたいときに，入力ルーチンの前後でfep_off，fep_onを呼び出します．

fep_force_on，fep_force_off

機能	FEPの一時的オン
形式	void fep_force_on (void) ; void fep_force_off (void) ;
説明	FEPの変換状態を一時的にオンにして漢字を入力させたいときに，入力ルーチンの前後でfep_force_on，fep_force_offを呼び出します．

※FEPCTRLの詳細については付属のドキュメントを参考にしてください．また，作者自身による解説がCマガジン誌91年3月号に掲載されています．

リスト5

```
        es.type = EDT_KANA | EDT_KANJI | EDT_USEFEP;
```

表2　入力文字種とFEPの制御のためのシンボル

EDT_DIGIT	半角数字
EDT_ALPHA	半角英字
EDT_KANA	半角カナ
EDT_SPACE	半角スペース
EDT_OTHER	その他の半角文字
EDT_HANKAKU	すべての半角文字
EDT_K_DIGIT	全角数字
EDT_K_ALPHA	全角英字
EDT_K_KATA	全角カタカナ
EDT_K_HIRA	全角ひらがな
EDT_K_KIGOU	全角記号
EDT_K_SPACE	全角スペース
EDT_K_OTHER	その他の全角文字
EDT_KANJI	すべての全角文字
EDT_ALL	すべての文字
EDT_USEFEP	FEPをオンにして入力する
EDT_NOUSEFEP	FEPをオフにして入力する

typeに指定できるシンボルを表2に示します．これらを組み合わせて使う場合は，｜（論理和）演算子を使います．制限したい文字種の組み合わせがこの表にない場合はどうするのでしょう．そのときは，kbdinから受け取った文字を自分で判断して適当でないものはeditstrに渡さなければよいのです．

簡易エディタの作成

文字列を入力するためのeditstrは，文字列入力に必要な機能がほぼそろったといってよいでしょう．そこで，このeditstrを使って簡易エディタを作成してみました(リスト6)．ここで作成するエディタはごく基本的な機能しかもっていませんが，文字列の編集ルーチンを使えば比較的簡単にエディタのようなものが作成できるということがおわかりいただけると思います．

このエディタの編集コマンドを表3に示します．

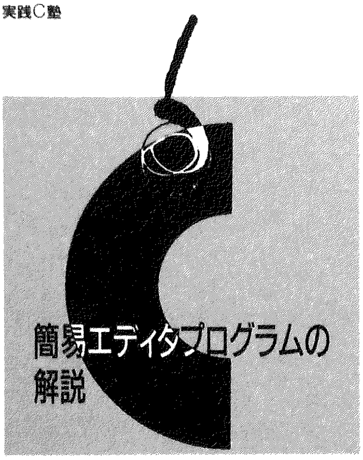
リスト6　(edit.c)

```
/*
 *  edit.c: 簡易エディタ
 *      written by Motohisa Ohno
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <time.h>
#include <io.h>
#include <jctype.h>
#include <jstring.h>
#include "kbd.h"
#include "tvram.h"
#include "editstr.h"

#define MAXLINE 1000    /* 最大行数 */
#define LINECHR 200     /* 1行の最大文字数 */

char *editfile;         /* 編集ファイル名 */
char *edptr[MAXLINE];   /* 行バッファへのポインタ */
int lnums;              /* 行数 */
extern int edit_attr;   /* 編集テキストの文字属性 */
extern int insert_mode; /* 挿入モード */

/*
 *  エラーメッセージを表示し、プログラムを終了する
 */
void error(char *fmt, ...)
{
    va_list ap;

    va_start(ap, fmt);
    vfprintf(stderr, fmt, ap);
    va_end(ap);
    exit(1);
}

/*
 *  ファイルから編集バッファに読み込む
 */
void ReadFile(char *fname)
{
    char buf[BUFSIZ];
    int n;
    FILE *fp;

    fp = fopen(fname, "r");
    if (fp == NULL) {
        fp = fopen(fname, "w");
        if (fp == NULL)
            error("can't open %s¥n", fname);
        lnums = 0;
    } else {
        for (lnums = 0; fgets(buf, sizeof buf, fp); lnums++) {
            n = strlen(buf);
            if (n && buf[n-1] == '¥n')
                buf[n-1] = 0;
            edptr[lnums] = strdup(buf);
            if (edptr[lnums] == NULL)
                error("out of memory (line#%d)¥n", lnums);
        }
    }
    edptr[lnums++] = strdup("");
    fclose(fp);
}

/*
 *  編集バッファからファイルへ書き込む
 */
void WriteFile(char *fname)
{
    int n;
    FILE *fp;

    fp = fopen(fname, "w");
    if (fp == NULL)
        error("can't open %s¥n", fname);
    for (n = 0; n < lnums-1; n++)
        fprintf(fp, "%s¥n", edptr[n]);
    if (edptr[n][0] != '¥0')
        fprintf(fp, "%s¥n", edptr[n]);
    fclose(fp);
}

/*
 *  画面の初期化
 */
void scrinit(void)
{
    printf("¥033[>3l");      /* set 25 lines */
    printf("¥033[2J");       /* clear screen */
}

/*
 *  編集バッファの内容を画面に表示
 *      lstart: 表示を開始する行番号
 *      offset: 左にスクロールアウトした桁数
 */
void dspscr(int lstart, int offset)
{
    int i;

    for (i = 0; i < 23; i++) {
        if ((lstart + i) < lnums && offset < strlen(edptr[lstart+i]))
            dspstr(0, i, 80, edptr[lstart+i] + offset, edit_attr);
        else
            dspstr(0, i, 80, "", edit_attr);
    }
}

/*
 *  入力待ちの間は、時間を表示する
 */
void dsptime(void)
{
    static char buf[20];
    time_t sec;
    struct tm *now;

    time(&sec);
    now = localtime(&sec);
    strftime(buf, sizeof buf, "%y-%m-%d  %H:%M:%S", now);
    dspstr(40, 23, 20, buf, edit_attr);
}

/*
 *  編集
 */
void Edit(void)
{
    static char buf[LINECHR];   /* 編集バッファ */
    char pbuf[20];              /* カーソル位置を表示するためのバッファ */
    int topln, offset;          /* 表示開始オフセット */
    int curln, curcol, lastln;  /* 編集位置 */
    int i, n;
    unsigned code;              /* 入力文字コード */
    EDITSTR es;                 /* 文字列を編集するための構造体 */

    /* データの読み込み */
    ReadFile(editfile);
    /* 入力待ちの間に実行する関数を設定 */
    setidlefunc(dsptime);

    topln = offset = 0;
    curln = curcol = 0;
```


ワンラインNEWS
東芝は高速な256KビットスタティックRAMを開発．8月からサンプル出荷を開始する．アクセス速度は10ナノ秒と業界最高水準を達成している．

```
    for (;;) {
        /* 文字列を編集するための構造体を設定する */
        es.lx = 0;
        es.ly = curln - topln;
        es.width = 80;
        es.length = LINECHR;
        strcpy(buf, edptr[curln]);
        es.bufp = buf;
        es.histlen = 0;
        es.histp = NULL;
        es.type = EDT_ALL;
        /* 文字列入力のための初期化 */
        editstr(EDS_BEGIN, &es);
        /* スクロールアウト領域にカーソルがないかどうか調整する */
        if (strlen(es.bufp) < curcol)
            curcol = strlen(es.bufp);
        else if (nthctype(es.bufp, curcol) == CT_KJ2)
            --curcol;
        es.curp = curcol;
        if (curcol < offset) offset = curcol;
        es.offset = offset;
        /* 位置を補正するために空文字を入力ルーチンに渡す */
        editstr(KBD_NULL, &es);
        curcol = es.curp;
        offset = es.offset;
        /* 画面全体の描画 */
        dspscr(topln, offset);
        for (;;) {
            lastln = curln;
            /* カーソル位置を画面左下に表示 */
            sprintf(pbuf, "%3d:%3d  %s",
                curln+1, es.curp+1, insert_mode ? "INS" : "OVR");
            dspstr(4, 23, 15, pbuf, edit_attr);
            /* 文字入力 */
            code = kbdin();
            /* 文字コード変換 */
            switch (code) {
            case KBD_CTRL_S: code = KBD_LEFT; break;
            case KBD_CTRL_D: code = KBD_RIGHT; break;
            case KBD_CTRL_E: code = KBD_UP; break;
            case KBD_CTRL_X: code = KBD_DOWN; break;
            case KBD_CTRL_A: code = KBD_HOME; break;
            case KBD_CTRL_F: code = KBD_CLR; break;
            case KBD_CTRL_R: code = KBD_RLDN; break;
            case KBD_CTRL_C: code = KBD_RLUP; break;
            case KBD_CTRL_M: code = KBD_CR; break;
            case KBD_CTRL_G: code = KBD_DEL; break;
            case KBD_CTRL_O: code = KBD_INS; break;
            }
            switch (code) {
            case KBD_RLUP:      /* [ROLLUP] = スクロールアップ */
                topln += 23;
                if (topln >= lnums) topln = lnums - 1;
                curln += 23;
                break;
            case KBD_RLDN:      /* [ROLLDOWN] = スクロールダウン */
                topln -= 23;
                if (topln < 0) topln = 0;
                curln -= 23;
                break;
            case KBD_UP:        /* [↑] = 1行上へ移動 */
                --curln;
                break;
            case KBD_DOWN:      /* [↓] = 1行下へ移動 */
                ++curln;
                break;
            case KBD_CR:        /* 改行の入力 */
                if (insert_mode && lnums < MAXLINE) {
                    /* 挿入モードであれば、行を分割する */
                    for (i = lnums; i > curln; i--)
                        edptr[i] = edptr[i-1];
                    edptr[i+1] = strdup(es.bufp + es.curp);
                    edptr[i] = strdup("");  /* dummy */
                    es.bufp[es.curp] = 0;
                    ++lnums;
                }
                /* 次行の先頭へ移動 */
                es.curp = 0;
                ++curln;
                break;
            case KBD_ESC:
                /* ignore [ESC] */
                continue;
            case KBD_BS:
                if (es.curp || curln == 0) {
                    /* 行頭以外か先頭行であれば、1行文字列編集 */
                    editstr(KBD_BS, &es);
                    if (es.offset != offset) break;
                    continue;
                }
                /* 先頭行以外の行頭であれば、前行と連結する */
                free(edptr[curln]);
                edptr[curln] = strdup(es.bufp);
                strcpy(buf, edptr[curln-1]);
                n = strlen(buf);
                if (n < (LINECHR-1))
                    strncpy(buf + n, edptr[curln], LINECHR - n - 1);
                free(edptr[curln]);
                buf[LINECHR-1] = 0;
                --curln;
                --lnums;
                lastln = curln;
                edptr[curln] = strdup("");
                for (i = curln+1; i < lnums; i++)
                    edptr[i] = edptr[i+1];
                es.curp = n;
                break;
            case KBD_CTRL_Y:
                /* 1行削除 */
                if (lnums == 0) continue;
                --lnums;
                free(edptr[curln]);
                for (i = curln; i < lnums; i++)
                    edptr[i] = edptr[i+1];
                if (lnums == 0)
                    edptr[lnums++] = strdup("");
                if (curln >= lnums)
                    --curln;
                lastln = curln;
                strcpy(es.bufp, edptr[curln]);
                break;
            case KBD_F10:
            case KBD_F5:
                /* 編集の終了 */
                break;
            default:
                editstr(code, &es);
                if (es.offset != offset) break;
                continue;
            }
            break;
        }
        /* 編集バッファの更新 */
        free(edptr[lastln]);
        edptr[lastln] = strdup(es.bufp);
        /* カーソル位置の補正 */
        if (curln < 0) curln = 0;
        if (curln >= lnums) curln = lnums - 1;
        if (curln < topln) topln = curln;
        if (curln > (topln + 22)) topln = curln - 22;
        curcol = es.curp;
        offset = es.offset;
        switch (code) {
        case KBD_F10:       /* [F10] = ファイルへ書き込み終了 */
            WriteFile(editfile);
            break;
        case KBD_F5:        /* [F5] = ファイルへ書き込まずに終了 */
            break;
        default:
            continue;
        }
        break;
    }

    /* 入力待ちの間に実行する関数の設定を解除 */
    setidlefunc(NULL);
}

/*
 *  編集ファイル名の入力
 */
char *getfilename(void)
{
    static char fname[80];
    static char fnhist[200] = "¥xFF";
    EDITSTR es;
    unsigned code;

    /* ファイル名の入力を促す */
    dspstr(0, 0, 12, "File Name : ", edit_attr);
    fname[0] = 0;
    es.lx = 12;
    es.ly = 0;
    es.width = 40;
    es.length = 80;
    es.bufp = fname;
    es.histlen = sizeof fnhist;
    es.histp = fnhist;
    es.type = EDT_ALPHA | EDT_DIGIT | EDT_NOUSEFEP;
    editstr(EDS_BEGIN, &es);
    do {
        code = kbdin();
        editstr(code, &es);
    } while (code != KBD_ESC && code != KBD_CR);
    return code == KBD_ESC ? NULL : fname;
}

int main(int argc, char **argv)
{
    /* 初期化 */
    tvraminit();
    kbdinit();
    fep_init();

    setfkey(5, "放棄");
    setfkey(10, "保存");

    if (argc > 1) {
        /* コマンドライン引数が指定されたときは、そのファイルを編集 */
        scrinit();
        editfile = argv[1];
        Edit();
    } else {
        /* コマンドライン引数が指定されなかったらファイル名を入力 */
        for (;;) {
            scrinit();
            editfile = getfilename();
            if (editfile == NULL) break;
            Edit();
        }
    }

    /* 終了処理 */
    fep_term();
    kbdterm();
    tvramterm();
    return 0;
}
```

ワンラインNEWS
日本電気HEはCD-ROMプレイヤーと一体型の家庭用テレビゲーム機「PC-Engine Duo」を開発した。RAM容量を従来機の4倍にあたる4Mビットに増やした。



簡易エディタプログラムの解説

表3　編集コマンド

キー	機能
[←], CTRL-S	カーソルを左へ移動
[→], CTRL-D	カーソルを右へ移動
[↑], CTRL-E	カーソルを上へ移動
[↓], CTRL-X	カーソルを下へ移動
[HOME], CTRL-A	カーソルを行頭へ移動
[SHIFT] + [HOME], CTRL-F	カーソルを行末へ移動
[ROLLUP], CTRL-C	次画面へ移動
[ROLLDOWN], CTRL-R	前画面へ移動
[DEL], CTRL-G	カーソル位置の文字削除
[BS], CTRL-H	カーソル位置の左にある文字を削除
[INS], CTRL-O	挿入モードの切り換え
CTRL-Y	1行削除
[RETURN], CTRL-M	改行
[F10]	ファイルへ書き込み終了
[F5]	ファイルへ書き込まずに終了

このエディタは，オンメモリのエディタです．

コマンドライン引数を指定せずにエディタ(edit)を起動すると，ファイル名の入力が促されます．コマンドライン引数を指定しないと，1つのファイルの編集を終了するたびに次に編集するファイル名の入力が求められますが，ここではヒストリー機能を使うことができます．ファイル名として入力できる文字は英数字のみで，FEPがオンのときは一時的にオフ状態になります．ここでESCを押すと，エディタは終了します．コマンドライン引数を指定すると，そのファイル名が編集ファイル名として設定されます．

ファイル名が与えられると，編集のための関数Editが呼び出されます．Editは，まずファイルから編集バッファにデータを読み込むため関数ReadFileを呼び出します．ReadFileは，ファイルが存在すればそのファイルからデータを読み込みます．ファイルを読み込むときは，行末の改行コードは削除されます．ファイルが存在しなければ，与えられたファイル名でファイルを作成してみて成功すればそのままファイルをクローズして戻ります．ファイル名として不適当な文字が指定された場合など，ファイルの作成に失敗した場合は異常終了します．

編集バッファは，文字列へのポインタの配列の形をとっています．1行ごとに必要なヒープメモリを確保して文字列の内容をコピーしていきます．文字列をヒープメモリに展開するためにはstrdupが使われています．strdupは，mallocとstrcpyを組み合わせたような関数ですが，このほかにもメモリ管理についてはCの標準関数をそのまま利用しており，エラーチェックはほとんどされていません．もともと実用目的のものではないので，実際のアプリケーションに応用する場合は，strdupが間接的に使用しているmallocなどのメモリ割り当て関数やそれを利用している部分で，きちんとしたエラー処理を行わなければなりません．

このエディタは，仕様としては1行につき200文字まで，全体で1000行まで扱うことができますが，大きなファイルを読み込ませることは暴走の原因となります．スモールモデルでコンパイルした場合は，編集できるファイルの上限はせいぜい40～50KBと考えてよいでしょう．もっとも，これだけ大きなファイルを編集しようとするとこのエディタにとってはかなりの負担となります．エディタの都合上，まったく空の(0行の)テキストを扱うことはできないので，ファイルが空のときは最後に空文字列を入れておきます．

次に，setidlefuncで時間を表示する関数dsptimeを設定しています．setidlefuncは，キー入力待ちの間に実行する関数を設定するものです．DOSアプリケーションの多くは入力待ちで無駄な時間を費やすものが多いものです．ここではその空き時間を利用して画面の右下に現在の時刻を表示させようというわけです．

次は，いよいよ編集のメインルーチンです．ここでは，2重のループが組まれています．基本的に，外側のループが行の移動などをあらわし，内側のループは文字列の編集のための（いつもの）ループです．ただし，文字列を編集するだけでも画面全体を処理しなければならないときは外側のループに抜け出すことがあります．

ここで，このループの中で使われているちょっとトリッキーな手法について説明しなければなりません．このループは，基本的にfor(;;)の形の無限ループです．また，それぞれのループの

ワンラインNEWS
日本DECはRISC型ワークステーション「DECstation5000/100シリーズ」を2モデル発売した．OSにUNIXを採用し「ACE」アーキテクチャに準拠している．

中でswitchブロックによる分岐処理が行われています．forブロックから抜け出すためにはbreak文を使いますが，同様にswitchブロック中の各case文の処理から抜け出すためにもbreak文を使います．これに対し，continue文はswitchブロックとは無関係でswitchブロックの中からcontinue;を実行するとswitchブロックの先頭ではなく，その外側の繰り返しブロック(ここではforブロック）の残りの部分をスキップして次の繰り返しに移行します．

ときとして，forブロックの中にあるswitchブロックのcase文から，直接forブロックを抜け出したいと思うことがあります．しかし，switchブロックの抜け出しとforブロックの抜け出しは，ともにbreak文を使わなければならないため，通常はリスト7のようなプログラミングをしなければならないでしょう．

私は，特別にgoto文を嫌っているというわけでもありませんし，よけいなフラグ変数を用意するくらいならばgoto文を使ったほうがよいと思っていますが，goto文を使用しなくてもよいのならそうしたほうがよいでしょう．リスト7の例は，リスト8のように置き換えることができます．

つまり，switchブロック内でのbreakとcontinueの適用範囲が異なることを利用して，switchブロックから直接forブロックを抜け出せるようにしたわけです．正直なところ，このほうがわかりやすいといえるかどうかは微妙です．もともと，continueはあまり使われない命令ですし，このようにブロックからの抜け出しが2段階になってしまうと，プログラムを読む人によってはかえってわかり難いかもしれません．とりあえず，プログラミング手法の1つと考えてください．

リスト7

```
    for (;;) {
      switch (cond) {
      case label1:
        ...
        goto skip;
      case label2:
        ...
        break;
      }
    }
  skip:;
    ...
```

リスト8

```
    for (;;) {
      switch (cond) {
      case label1:
        ...
        break;
      case label2:
        ...
        continue;
      }
      break;
    }
```

さて，話はエディタのメインルーチンに戻ります．外側のループの先頭では，文字列を編集するための構造体を初期化しています．編集位置はカーソル行の左端から80桁，編集バッファは内部変数bufを使います．編集バッファにはあらかじめ現在行の内容をコピーしておきます．ヒストリー機能は使わず，すべての文字種を入力の対象とします．カーソル位置が文字列よりも後ろにあったり，編集行の文字列がスクロールアウトして表示領域に入らなくなってしまってはいけないので，いったん文字列入力の初期化を行なったあとで再度構造体のメンバを調整しています．dspscrは，編集バッファの内容を画面全体に表示する関数で，表示を開始する行番号と左へスクロールアウトする桁数を指定します．

画面全体を表示し1行編集の準備ができたら，1行編集のループへ移行します．まず，カーソル位置を表示して文字入力へ移行します．この関数全体の処理が遅いので，かなり実行速度の速い機種でないと画面をスクロールしたときに惰性が生じてしまうため，文字入力の前にキーバッファをクリアしたいところです．しかし，FEPで漢字入力をすると2文字目以降が無視されてしまうのでこれは諦めます．

文字入力直後のswitchブロックは，CTRLと英字の組み合わせでカーソル移動を行うことができるようにキーコードを変換するためのものです．基本的には，ダイヤモンドカーソルに似せて変換していますが，文字入力関数には単語移動の機能がないのでCTRL-AやCTRL-Fは行頭・行末へ移動する機能が割り当てられています．もし，CTRL-Q+キーなどの2ストロークキーに機能を割り当てたい場合は，case KBD_CTRL_Q:文を記述してさらにkbdinを呼び出して分岐処理すればよいでしょう．ただ，このエディタは2ストロークまで使わなければいけないほど多くの機能はもっていません．

次のswitchブロックでは，各機能を実際に処理しています．編集行が移動したり編集文字列が左右スクロールする場合は，編集バッファに現在の編集

内容を割り当てたり，画面全体を再描画するために外側のループへ移行しなければなりません．ここでは，前述のbreakとcontinueに関する手法を使っています．

上下のカーソルキーや［ROLLUP］［ROLLDOWN］キーが押されたら，行移動のための処理を行います．リターンやバックスペースは，そのままeditstrには渡しません．たとえば，挿入モードがオンであればリターンを押された位置でその行を分割します．挿入モードがオフのときは，次行の先頭に移動します．行頭でバックスペースが押された場合は，現在の行を前行の最後に連結します．行頭でなければeditstrに渡して文字列編集のために使います．ESCが押されても文字列入力を終わらせる必要はありませんから，editstrに渡しません．CTRL-Yが押されたら，その行を削除します．［F5］と［F10］は，編集を終了させるためのキーで［F5］はファイルへの書き込みを行わず，［F10］は書き込みを行います．なお，ファイルへ書き込む際は，読み込むときに削除した行末の改行文字を追加します．

このループから抜け出したら，行の移動などがあったわけですから，現在編集中の文字列を編集バッファに反映させなければなりません．編集バッファの現在行のために確保されているメモリを解放し，編集中の文字列をヒープに確保したメモリにコピーして編集バッファの指すポインタに設定します．

外側のループを抜け出せるのは，［F5］か［F10］が押されたときだけです．それ以外のキーが入力されたときは，ループの先頭に戻ります．外側のループから抜けたら，入力待ちの間に実行する関数の設定を解除してEditから戻ります．

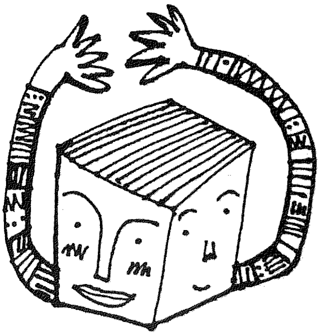
最後に

入力ルーチンに関する説明は以上です．さらなる改良については読者の方々におまかせします．ここで作成したeditstr関数をいろいろと改良してみるのもよいですし，新たなルーチンを組んでみるのもよいでしょう．不要な機能を取り除いて，もっと軽い関数にするということも考えられます．文字列の入力はさまざまな場面で必要となりますから，一度作成しておけばさまざまなアプリケーションで応用できるでしょう．

おまけ

7月号で紹介した「コマンドライン引数のすべてを1文だけで表示してください」という問題について，PC－VANでFlowerField氏からリスト9のようなプログラムをご教示いただきました．

確かに，セミコロンは1個だけしか使われていませんし，コマンドライン引数のすべてを表示しています．FlowerField氏に感謝するとともに，自らの勉強不足を反省しているしだいです．



リスト9

```
#include <stdio.h>

void main(int argc, char **argv)
{
    ++argv, --argc > 0
    ? printf("%s ", *argv), main(argc, argv)
    : putchar('¥n');
}
```


ワンラインNEWS
日本電算機は超小型，高速のXウィンドウ端末「Xface」を発売した．CPUとディスプレイを一体化，奥行き13.6cmと薄型設計にした．

98活用アセンブラ技法

番外編 第1回

アセンブラ命令コードの把握

星野 操

前回までで，画面アクセス強化のための一連のプログラム作成は一区切りとなりました．その過程でマクロの作成やその使い方，PC-9801の外字の使用法や常駐プログラムの作成方法，そのほかの技術などについて解説してきました．今回から数回，アセンブラやそのデバッグに関してのよもやま話をしていこうと思います．

アセンブラの処理系

86系のアセンブラ処理系といえば，やはりMASM（Microsoft Macro Assembler）とTASM（Turbo Assembler)になってしまうでしょう．高速なアセンブル，マルチパスアセンブルによるジャンプ命令の最適化などを売り物にしていたOPTASMというのもありましたが，最近はあまり名前を聞くこともなくなりました．アセンブルスピードではTASMに勝っているものの，やはりTASMが出てきてからはもうひとつぱっとしなくなってしまったようです．

86系のCPUはもちろんインテル社が開発したチップですから，まず最初にintelがこれらのCPUのための各種処理系を開発しています．これらはintelの開発環境であるiRMXなどの上で動くほか，一種のシステムコールのシミュレーターを介してDOSなどの上でも走らせることができます．

インテル社の提供した処理系には，アセンブラやPL/Mなどがありますが，やはり現在いちばん一般的な開発環境はDOSでしょうし，いちばん一般的な処理系はDOSの上で動くこれら2つの処理系ということになるでしょう．

TASMとMASMの互換性

TASMには，Idealモードという独特の記述方法がありますが，それを除けばこれらは基本的にはインテルのアセンブラと同様な記法を採用しています．ただし，マクロそのほかの記述はインテルのものとは大きく異なっています．TASMはMASMを包括しようとしていますので，MASM的なモードで使っていれば，見かけ上さほどの違いは存在しないことになります．

もっとも実際に使ってみると，些細な差がいくつも出てきます．バージョンにもよりますが，連載中にひっかかったPROCの引数マクロのスコープや予約語の差異とそれを再定義したときの挙動，インデックスドオペランドの記法などがそれにあたります．TASMには，より厳密にMASMをシミュレートするモードもあるのですが，それとても完全とはいえません．実際にクリティカルな部分を使用する場合には，あらかじめコンパチビリティをチェックしたほうがよいでしょう．

とくに，パスの問題については注意が必要です．MASMは基本的な2パスアセンブラですから，1パス目でアドレスを計算し，2パス目でコードを実際に生成します．このため，if1／if2などという命令が用意されており，ダミーの命令を作成するときに使用できました．

たとえば，拡張命令などでアドレスシンボルを参照する際，単にdwなどで定義すると1パス目に未定義でエラーになることがあります．このような場合，if1では仮に“dw 0”などでダミーの命令を生成し（アドレスだけ合わせて)，実際にシンボルが生成された2パ

ス目でdw命令を使用することができます．

ところが，TASMは基本的に１パスアセンブラです．このため，未定義のラベルを参照していたところは，実際に定義された時点でそれを「書き直し」ます．ですから，if1／if2命令はMASMのような意味をもちません．そのかわり，直接未定義シンボルをdwなどの引数に取っても最後までエラーにはなりません．このような命令を利用する場合には，注意する必要があります．

また，１パス目でマクロを定義し，２パス目でそれを利用する（１パス目では同一サイズのスタブを置いておく）というようなプログラムもTASMでは通りません．

さらに，最近のTASMでは今度はマルチパスをサポートしています．アセンブラでは，前方参照をしている場合に，どういうコードを生成したらよいか厳密にはわからないので，適当に冗長な命令を生成します．実際に生成されたコードが仮定されたものよりも短かった場合には，NOP命令などがつめられます．マルチパスを利用すると，それらのオプティマイズが可能になります．さらに，条件ジャンプのレンジをオーバーしたときに補条件の条件ジャンプと無条件ジャンプに置き換える最適化も可能となります．これらは，OPTASMで実現されていたものですが，このモードを利用した場合でも，やはりif1／if2の意味はMASMとコンパチブルにはなりえません．

このほかにも，TASMの独自拡張されている部分は少なからず存在しています．TASMがMASM 5.xに新しい機能をつけ加えていき，さらにMASMも新しいものへと進んでいくと，5.x以降の機能の互換性は低くなるかもしれません．

以上のような点に注意して，これらの処理系を使う必要が出てくるでしょう．

マクロ

アセンブラが大きく機能強化されたのは，MASMの5.0からだと思います．それ以前のMASM，とくに1.x〜3.xでは，仕様の変更はほとんどなかったといってかまわないでしょう．主としてバグとりが行われたという感じでした．

MASMの5.xから，簡易セグメント指定と高級言語インターフェースが導入されました．さらに，substr／catstrなどの文字列操作用のマクロなどが追加され，その気になればそれこそ「なんでもできる」ようになってしまいました．また，現在でもいろいろな機能の強化がなされています．

こうした多様な機能の中でも，とくにマクロについてもう一度触れてみたいと思います．

本連載でも多用した非常に強力なこれらの処理系のマクロですが，インテル純正のそれとは大きく異なっています．これらは，もともとCP/M 80の上で動くM80時代に実装されていたものをもとにしています．

最近実装されたsubstrなどの文字操作マクロなどは別にしても，MASM系のマクロは非常に強力です．これは，以前に書いたように「マクロ定義マクロ」が記述できることが１つの大きな理由です．また，irpやreptなどの繰り返しをマクロがサポートしているというのも大きなメリットです．

CPPのようなプリプロセッサでもある程度のマクロ展開はできます．しかし，これは定義語を定義することができません．メタマクロというものが存在していないのです．また，マクロの展開も基本的に一方向で限定されたものです．複雑な構造を含ませることはできませんし，マクロの中で条件コンパイル（アセンブル）をサポートすることすら不可能です．もちろん，これだけでも実用面でのメリットは十二分にあります．しかし，本連載で使用したような構造マクロや関数マクロなどを定義しようと思うとまったく無力です．

さらに大きいメリットは，マクロの実行中に「アドレス値」が使用できるということです．内容はよく知りませんが，M4などのメタ言語と呼ばれるものは，CPP以上に強力なマクロ言語であると思われます．しかし，これらの強力なマクロ言語よりもMASM系のマクロが優っているのは，マクロ内の制御構造などでアドレス値を使用できるからです．

たとえば，構造マクロのagainの記述のように，特定ラベルとカレントアドレスとの差の大きさによって，生成するコードを変化させるというようなことが可能です．もしも，同様なことをマクロ言語でやろうと思うと，アセンブラが生成するコードサイズを知り，それによってアドレスを計算しなければいけません．このアドレス管理に要する手間というのは半端ではありませんから，それを組み込みでやってくれるというMASM系のマクロはたいへん便利なのです．

MASMなどのマクロだけで，クロスアセンブラを作成することも可能です．アドレッシングモードなどには制限が生じますが，基本的に各命令コードと同一の名称のマクロを作成し，それらがDB命令などで命令コードやオペランドを生成するようにすればよい

ワンラインNEWS
YHPは100万円を切る低価格ワークステーション「HP Apollo 9000シリーズ400」の新機種を発売した．卓上型の「モデル425t CRX」など５モデルがある．

わけです．このようにして作成されたクロスアセンブラは，MASMとまったく同様にアドレスを参照した条件判断などを使用することができ，下手をすると既存のアセンブラより強力になるぐらいです．

もちろん，マクロをあまり多用するとスピードが遅くなりますし，あまり大きなプログラムになると「わけがわからない」エラーが出てきます．前回のプログラムも，最初はリンクせずに，全部インクルードさせて1つのファイルにしてアセンブルしていましたが，TASMではわけのわからないエラーになって（ストリングスペースがたりないというエラー，オプションで指定しても効果がない）アセンブルできなくなってしまいました．

やはり，あまり大きなプログラムにしてしまうのは避けて，リンクをしたほうがよいでしょう．構造化マクロなどを利用している場合には，とくにこのような配慮が必要です．

アセンブラの命令

1つのCPUのアセンブラを理解していると，ほかのCPUのアセンブラも理解しやすくなります．もちろん，RISC CPUなどのように大きくアーキテクチャが変わるようなCPUだと多少話が違ってきますが，たとえば8086のアセンブラを理解していれば，Z80や68KぐらいのCPUであれば，そのアセンブラを把握するのは困難ではないでしょう．

これは，これらのCPUがもっている命令セットにそれほどの差異がないからです．これらのCPUは，いずれも加算命令，減算命令，論理演算命令，分岐命令など仔細は異なりますが，基本的には同一のコンセプトをもつ命令群をもっています．なぜ異なるCPUに同様な命令セットが用意されているのか？　それを理解するためには，基本的なハードウェアの知識が必要になります．

CPUを構成するうえで絶対に不可欠であるという命令はさほどありません．なんでも，ある1つの動作をする命令（詳細は忘れましたが）だけで，そのほかのすべての命令を記述することが可能なのだそうです．もちろん，このようなアーキテクチャをとることは効率的ではありません．ある目的を達成するための記述が，あまりにも複雑になってしまうからです．

実際には，いくつかの動作ユニットを選択してインプリメントする必要があります．選択される動作ユニットは，「人間が理解しやすいもの」であり，なおかつ「機械が理解しやすいもの」でなくてはいけません．後者は，「ハードウェア的に実装しやすいもの」につながります．これらは，基本的に相反する方向を差しています．既存のCPUに搭載されている命令セットは，その妥協点であるわけです．その妥協点を理解するためには，とくに後者についての理解，すなわち「CPUの立場に立った」理解が必要なのです．前者については，高級言語などのトランスレータによって補うことができます．

この「CPUの立場に立った」視点というのは，いろいろな場面で要求されます．高級言語の仕様などでも，「なぜそれがそうなっているのか」というのは，プログラムを実行する側，CPUの側に立ってみればなるほど（しかたがないか）と思われることが多いものです．

たとえば，8ビットの二項演算を考えてみましょう．この演算というのは，極論すれば16ビットの入力と8ビットの出力との間の一種のマッピング動作といえます．それがどんな複雑な演算だったとしても，このマッピングさえきちんとできれば演算は実行できます．つまりは，64KBのROMが一個あれば，どのような演算でも表現できるのです．

しかし，このような過剰に「一般的な」マッピングをすることは，チップ面積の限られているCPUでは得策ではありません．そのため，ハード的に「なるべく簡単に」表現できる命令セットが好んでインプリメントされるのです．このような，簡単な演算であれば実行も高速であるし，CPUの実行「単位」を構成するにはふさわしい，ということになります．

ワンラインNEWS
リコーはワープロやパソコンで使うアウトラインフォントを作成するためのソフトを開発した．3次関数曲線の理論を採用し，一段と美しい曲線部を表現できる．

論理演算

このように命令を入力から出力へのマッピングと考えると，いちばん構成が簡単なのは「論理演算」です．NANDまたはNORさえあればすべての論理演算が構成可能であるのは明らかですが，実際には，NANDなどの命令は「人間の思考になじまない」ので，動作ユニットとしてインプリメントされることはまずありません．

たいていのCPUは，反転論理ではないAND／OR／XOR命令などをもっているでしょう．単項演算NOTもインプリメントされている場合がほとんどでしょうが，これは理屈上は定数－1とのXORと解釈することもできます．逆に，AND／OR／NOTがあればXORを構成することができます．これらに加えてNAND／NORなど，すべての論理演算をインプリメントするのも1つのやり方（人間には，そのほうが多少は便利）でしょうが，使用頻度の低い命令をユニットとしてもっているのは馬鹿馬鹿しいともいえます．実際にどこまでの命令をCPUがもっているかは，CPUの設計思想によります．コントローラーなどに使うようなCPUでは，あるいは反転論理まで命令セットに含めているかもしれません．

なぜ論理演算が簡単かというと，「1つの入力ビット」は「1つの出力ビット」に対してのみ影響を及ぼし，ほかには影響を及ぼさないからです．このため入出力ビット間の独立性が高く，命令を構成するユニットが非常に小さくなるからです．たとえば，AND命令をインプリメントするには，2入力のANDロジックが出力ビット数分だけあれば済む，という具合です．

算術演算

このようにCPUに取っては，論理演算がいちばん基礎となるものです．しかし，人間の実生活で演算といえば普通は算術演算です．論理演算命令から数値演算を構成することは可能ですが，これはANDとNOTからNANDを作る，というほど簡単ではありません．もっとも簡単な算術演算である加算であっても，複数ビットのものを論理演算だけで行おうとするのは困難です．ただし，加算も比較的単純な動作の繰り返しですから，ハードウェアで特定ビット幅の加算器を構成することは困難ではありません．

加算では，入力ビットが1ビット変化するだけでも，それが繰り上がりをひき起こして複数の出力ビットが変化することがあります．しかし，加算動作の単位について見てみると，ある桁の加算はそれに対応する桁の入力ビットと，その下の桁の「繰り上がり」だけを見ていれば構成できることがわかります．論理演算よりは多少複雑ですが，かなり近いレベルでハードウェアを構成できます．実際，TTLのICでも「加算器」というユニットが存在しています．

減算の複雑さも加算と同程度です．ですから，たいていのCPUには少なくとも加算と減算命令が用意されています．もっとも例外はあります．たとえばシングルチップの8748などには減算命令がありませんし，8080の16ビット演算も加算のみで減算はありません．このような場合は，減数の補数を加算することになりますが，NEG命令がインプリメントされていればそれを使うことができ，そうでなければビット反転してから1をたすという方法で補数を作成することができます（インクリメントについては後述）．

なお，加減算では多桁の演算を行う場合に繰り上がり・繰り下がりを考慮する必要があります．繰り上がりなどにはキャリービットが使用されますが，これを演算の際に考慮するかどうかの区別によって，加減算それぞれ2種類の命令が用意されたり（たとえば，ADDとADC），されなかったりします．もちろん1種類の命令しか用意されていない場合でも，キャリーのクリアとADC命令，またはADD命令とキャリー参照による条件分岐を利用した1の加算によって代用することが可能です．2種類ともそろえられているとしたら，それは人間の理解しやすさを優先したものといえるでしょう．

これに比較して，四則の中でも乗算除算，とくに除算は複雑さの度合いが高くなります．加算と同様に，1ビットの入力変化が複数の出力ビットに影響を及ぼしますが，その及ぼし方は，ほかの入力に依存し規則的ではありません．乗算除算をより基本的な動作ユニットに分解すると，シフトと加減算の繰り返しで表現されます．ですから，安いCPUではこれらの命令はユニットとしては実装されていませんし，実装されていたとしても（シフトと加算／減算を繰り返すので）実行速度が加減算に比べて遅くなってしまいます．

ただ，乗算についてはハード的な工夫により多少高速化を図ることができます．たとえば，4ビット×4ビットの乗算は，（前記のように）256バイトのマッピングROMで一気に結果を出すことができます．多ビット×多ビットの場合でも，4ビットごとに分割し，その（高速で演算される）部分積を加算することで，速度を上げることができます．もっとも，部分積の加算数はN^2のオーダーで増加しますが，シフ

ワンラインNEWS
キヤノンは日本電気や日本IBMのパソコンなどとプリンタが共有できるビジネスワープロ「キャノワード7500」「同6500」の2機種を発売した．

ト＋加算のアルゴリズムだと実行時間はNのオーダーで増加するだけですので，乗算するビット数が大きくなると不利になります．ほかにも，乗算専用のシフトレジスタなど，ハード機構を追加することで高速化を図ることができ，実際8086に比べて80286／V30などのCPUの乗算命令などは，クロック数レベルでもずいぶん高速化されています．

どのようなインプリメントが現実にされているかはわかりませんが，いずれにしろ乗除算命令は「人間に取ってわかりやすい（不可欠な）」命令ではあるけれど，「CPUにとっては複雑な」命令であり，インプリメントされていたとしても，内部的にはより小さな実行ユニットの組み合わせ・ループで構成されているので，ほかの命令に比べて非常に時間がかかるということに注意する必要があります．対象が浮動小数演算になると，さらにこの度合いがひどくなるというわけです．

シフト

乗除算命令でも使用される「シフト」は論理演算の一種といえますが，これも実行ユニットとして用意されている場合がほとんどです．シフトは，NOTと同様に二項演算ではなく単項演算に該当します．出力ビットは，入力ビットの１ビット（右ビットか左ビット）にのみ影響を受けますから，ハードウェアを構成するのは簡単です．TTLレベルで表現すれば，シフトレジスタを使用すればよいことになります．

シフト動作は，シフトアウトしたビットの処理とシフトインしたビットの処理で，算術シフトや論理シフト，ローテートなどの区別が生じることになりますが，基本的には大きな差異はありません．

このように，一般的なハードウェア構成だと，シフト動作は１ビット単位で行われることになります．多桁のビットシフトはユニットとして用意されていないか，あるいは内部的に１ビットシフトを繰り返し実行するために時間がかかることになります．

多ビットのシフトを高速に実行するためには，シフト幅に応じた入力を直接選択し，それを出力とすればよいわけです．このようなシフト機構をバレルシフタと呼びます．比較的大きなチップ面積を専有するというデメリットはありますが，80386のようにこのようなシフタをもっているCPUだと，多桁シフトでも速度が低下することはありません．

インクリメント，デクリメント

もう１つ，通常のCPUにはインクリメント，デクリメントという命令が実装されているのが普通です．これらは，CやModula-2のような高級言語にも導入されているほど一般的です．

中には，68000のように１だけではなく，小定数を加減算できるように一般化されているものもありますが，普通は１の加減算だけが特別扱いを受けているのです．これによって，インクリメント，デクリメントではその加算数を明示しなくてすむ（そのために，命令語長が短くなる）というメリットが生じます．

また，１の加減算というのはハード的にもカウンタ構成を取れば簡単に実装できますし，カウンタは加算器よりも簡単なハードウェアで構成できます．たとえば，各レジスタにおのおのインクリメント，デクリメントをするハードウェアを付加することによって，独立にこれらの処理を高速に行うことができるようになります．

以前のCPUでは，これらの「演算」を行える場所が限られていました．さすがに，プログラムカウンタのインクリメントぐらいは独自のハードウェアをもっていたでしょうが，相対ジャンプ命令などでアドレスを「計算」する必要がある場合には，レジスタの演算を行うのと同じユニット（ALU）を利用して演算していました．貴重な資源を時分割して使用していたわけです．

しかし，これではどうしても効率が悪くなります．そこで，最近のCPUではこれらの演算機能を必要に応じて多数配置しています．たとえば，「MOV AX, [BX+SI+2]」という命令を実行するためには，[BX+SI+2]がどのようなアドレスを指しているかを計算しなければいけません．8086では，これを汎用の演算ユニットを流用して行っていたため，アドレス計算に時間がかかりましたが，80286などの新しいCPUでは，専用のハードウェアを実装したため，複雑なオペランドを利用しても，ほとんどよけいな実行時間はかからなくなっています．

以上のように，CPUに実装されている「演算」関係の命令は，往々にしてハードウェアに対応づけが可能です．TTLなどのハードウェアユニットとして存在しているものと，命令のユニットとは，かなり相関があるということが理解できるでしょう．

転送系の命令

そのような演算命令に加えて，CPUには転送系の命令，そして分岐系の命令があります．ほとんどのCPUは，内部にレジスタをもっており，これに対して演算を行うようになっています．レジスタはCPUの内部にあることから，高速に操作できるためです．これを押し進めるとRISC的な構成になります．

高級言語とのインターフェースを考えると，メモリの内容もレジスタと同様に演算に使えるようにしたほうが便利であり，実際いわゆるCISCはその方向に進んできています．しかし，やはり可能なかぎり内部レジスタを使用することが，スピードとサイズの両面で有利であることは明らかです．レジスタは数が限られていますから，それを表現するためのビット数が少なくてすみ，結果として，アドレスをなんらかの形で表現しないといけないメモリオペランドをとるよりも，少ないビット数で命令を表現できるからです．

このようにレジスタは重要な意味をもっていますが，その内容は当然のことながらほかのメモリワーク・レジスタとやりとりができないといけません．このような内容の転送を行う命令が転送系の命令です．転送系の命令は，単なるコピー転送のほかに，特殊なものとして，スタック操作を伴うもの，交換系の命令があります．

実際，転送系の命令が使用される頻度は非常に高く，プログラムのほとんど（80%以上？）の時間は，転送系命令に使用されるという調査結果もあります．ですから，多くのCPUはこの転送系の命令が高速に使用できるように配慮してあります．たいていの場合，レジスタ⟷レジスタ間の転送命令は，最小のクロックで実行可能になっています．

スタック操作系命令の実行パフォーマンスは，call命令のそれをも支配しますし，とくにスタック渡しを多用する高級言語では実行パフォーマンスを大きく支配します．これらの命令はスタックポインタのインクリメント，デクリメント操作が絡むため，昔のCPUでは単なる転送よりもずいぶんとオーバーヘッドが大きかったのですが，80286クラスになると，単なる転送と同等までパフォーマンスが上がってきています．下手にメモリに値をセーブするよりも効率が高いぐらいです．

転送系では，あとLEA命令についてコメントしておきます．これは，引数にとったアドレッシングモードが指すアドレスをレジスタに格納するというものです．たとえば，変数のアドレスを引数としてサブルーチンに渡す場合などに有効です．もっとも，絶対アドレスをもつ変数のアドレスを計算する，というような場合には単なる定数の読み込みで十分です．実際に使用されるのは，スタック上のローカル変数のアドレスを取る場合や，配列など[BX+SI]などの形の変数アドレスを取得する場合などに限られるでしょう．

分岐系の命令

分岐系の命令には，無条件分岐，条件分岐，サブルーチンコールなどがあります．CPUによっては（たとえば8080など），条件コールや条件リターンなどが導入されていることもあります．

飛び先は，以前はダイレクトに飛び先アドレスを示すことが多かったのですが，最近では現在のプログラムカウンタの値からのオフセットを示す場合がほとんどです．以前は，プログラムカウンタに対する演算が必要になる（オーバーヘッドが大きくなる）こと，そしてハンドアセンブルなどをする場合に計算が厄介であったことなどから，絶対アドレスが多用されていました．

しかし現在では，相対ジャンプアドレスを利用すると飛び先アドレスが多くの場合1バイトで表現できること，またプログラムがリロケートされてもアドレスオフセットを変更する部分が少なくてすむことなどの理由により，相対アドレスが多用されるようになっています．86系のCPUでは，ジャンプ命令は基本的に相対アドレスです．ただし，テーブリングなどにより，レジスタ経由で飛び先を示す場合には直接アドレスになります．

8086では，条件ジャンプはすべて1バイトのオフセットしか取ることはできません．たいていの場合はこれで十分なのですが，このレンジをはずれてしまうと補条件の条件ジャンプと無条件ジャンプを組み合わせる必要があります．これが昔のMASMでは，非常に厄介な部分だったのですが，最初に名前の出たOPTASMや最近のTASMなども，これをシステムで自動的にサ

ワンラインNEWS
YHPはLANの標準規格のひとつである「10BASE-T」に対応したLAN製品を拡販する．専任のSEを置き，営業体制を強化するとともにユーザー向けのセミナーも開催．

ポートできるようになっています．

サブルーチンの呼び出しは，アドレスをスタックに残す以外は基本的には分岐と大差ありません．スタック操作の高速化にともない，サブルーチンの呼び出しも高速化されてきています．

CPUの主要な命令の概念は，ほぼ以上でカバーされるはずです．もっともこのほかに，多少のCPU固有な命令群が用意されているのが普通です．たとえば，8086系ならストリング命令関係などです．しかし，これらはある意味で応用であり，命令のマクロ化によって実行速度やコードサイズを改善するというようなものですから，基本を押さえておけば理解するのは困難ではありません．

アドレッシングモード

各CPUのアセンブラを理解するうえで，もう1つ大切なのはアドレッシングモードです．多くの命令では，レジスタ←→レジスタ間の操作にとどまらず，メモリとの間の演算も可能です．CPUによっては，メモリ←→メモリ間の演算をサポートしている場合もありますし，少なくともレジスタ←→メモリ間の転送命令はサポートされているはずです．

このように，主としてメモリ操作をするときに，対象となるメモリをどのように表現するかというのがアドレッシングモードなのです．ただし，普通にアドレッシングモードといえば，レジスタ，定数値などを表現するものも含めます．

8080などのCPUでは，メモリはきわめて限定的な使用しか許されていませんでした．スタック操作，レジスタとの転送などに加えて，オフセットなしのレジスタインデックスドアドレスのみが可能でした．これに比べて8086などでは，メモリを演算の対象（演算結果を格納する場所）に指定できるほか，そのアドレスを表現する方法も増えています．

つまり，直接アドレス，BP／BX／SI／DIインデックスレジスタと8/16ビットのオフセット指定，BP／BXとSI／DIの和に対するオフセット指定です．これらのレジスタの値と定数値との加算によって，対象となるアドレスを決定するというものです．

CPUによっては，メモリの内容に対してのインデックスアドレスを使用する（たとえば，2000番地の内容に5をたしたアドレスを，演算対象とする）ことも可能です．

こういうアドレッシングモードですが，BPを除くほかのレジスタについてはオフセット0のときにコードの短い記法があるのに，BPに対してはありません．このような非対称性を理解するためには，命令コードのビットマップについてある程度知っておく必要があります．

二項演算

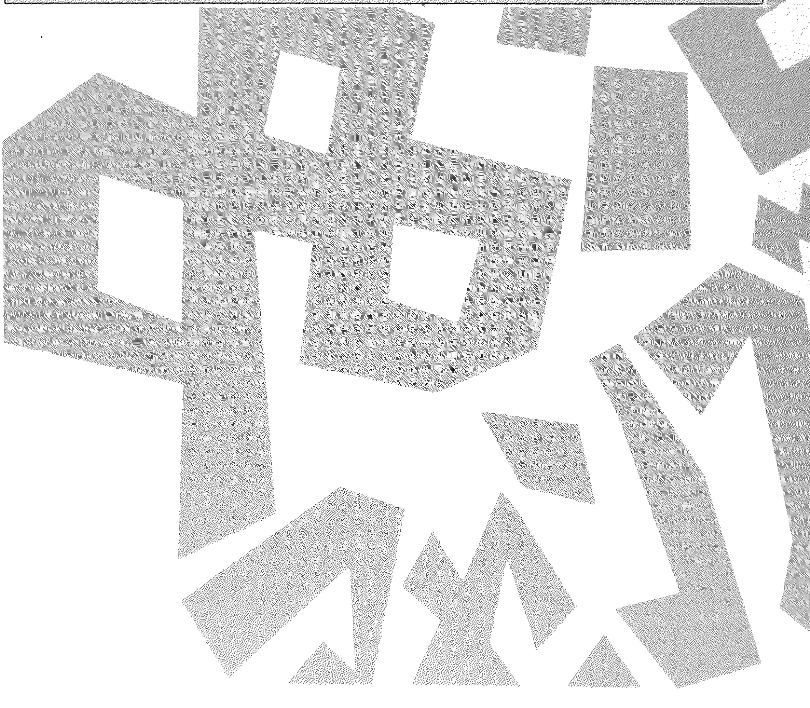
多くの二項演算（転送を含む）命令コードは，2バイト命令です．このあとに，必要に応じて，2バイトの絶対アドレスや8/16ビットのオフセット値が続きます．ですから，命令コードの長さは可変長です．

この2バイトのうち，後ろの1バイトがアドレッシングモードを示すバイトです．これは，図1のようにビットマップ表記されます．

上位2ビットがアドレッシングモードを示すフィールド，次の3ビットがレジスタを示すフィールド，最後の3ビットがレジスタまたはメモリのアドレッシングモードを示すフィールドで

図1

```
 7  6  5  4  3  2  1  0
|MOD|REG    |R/M     |
```



す．

REGフィールドは，0～7がワードの場合それぞれAX／CX／DX／BX／SP／BP／SI／DIの8個のレジスタに対応し，バイト演算の場合はAHL／CL／DL／BL／AH／CH／DH／BHに対応しています．ですから，BPなどのレジスタがバイトアクセスできないのは命令コードから考えても明らかです．

R/Mフィールドは，それがレジスタであると考えられる場合にはREGフィールドと同一の意味をもちます．一方，メモリアドレッシングであると考えられる場合には，0～7が［BX+SI］／［BX+DI］／［BP+SI］／［BP+DI］／［SI］／［DI］／［BP］／［BX］に対応します．デフォルトのセグメントレジスタは，BPが絡む場合のみSS:となります．これは，スタック内のローカル変数を，より効率的にアクセスするための仕掛でしょう．このように，アドレスレジスタの組み合わせが8つしか取れないことから，［SI+DI］のようなアドレッシングモードや［AX］など，アドレスレジスタ以外のレジスタを利用したアドレッシングができないことが理解できます．

MODフィールドは，そのR/Mフィールドを修飾するためのビットです．これが0の場合はオフセット値なしを意味します．当然，命令コードは短くなります．ただし，R/Mが6すなわち［BP］のアドレッシングモードのときのみ，これはアドレスの絶対指定と解釈され2バイトのアドレスがあとに続きます．1のときには8ビットの2のときには16ビットのオフセット値があとに続き，それが対象となるアドレスレジスタに加算されます．8ビットの場合は，符号拡張が行われます．このことから，オフセットが小さいほうが命令コードが短くなることがわかります．最後に，これが3だとR/Mフィールドはレジスタであると解釈されます．

このことからわかるように，8086の命令では，「レジスタ⟷レジスタ」間の演算が「レジスタ⟷メモリ」間の演算のみ可能で，「メモリ⟷メモリ」間の演算は不可能です．また，［BP］は結局［BP+0］と解釈され，［BX］などのアドレッシングよりも，命令コードのサイズが大きくなります．なぜこのようなことが生じているのかについても，ビットマップとその意味を理解すればわかりますし，存在しないアドレッシングモードを書いてアセンブラに怒られることもなくなります．

なお，80386以上のCPUの32ビットアドレッシングモードでは，これらの非対称性はほとんどなくなっています．その分，命令コードはどんな場合でも長くなってしまいます．昔から80系のCPUは，アドレッシングモードなどを制限し，非対称性を増しても命令コードを短くし，きちんとコーディングすれば短くて高速なプログラムを作成できることを目指していたといえます．これは，もともとこれらのCPUが「コントローラ」として開発されていたことに依存するものでしょう．

さて，二項演算のオペランドを示す方法はわかりましたが，これだけだとメモリとレジスタどちらが演算結果を格納する場所になるのかわかりませんし，演算がバイト演算なのかワード演算なのかもわかりません．

これらを示すビットは，命令コード2バイトの前1バイトの中に用意されています．一般に，バイト，ワードの区別のある命令の前1バイトの最下位ビット（wビット）はこれを示すビットです．さらに，その前の1ビット（dビット）が転送方向，すなわち演算結果を格納する方向を示すビットなります．

二項演算はMOV／ADD／ADC／SUB／SBB／AND／OR／XOR／CMPの9種類ありますから，これだけで命令コードの領域の9×4÷256と，1/4ぐらいを占めていることがわかります．

なお，AX／ALと絶対アドレスメモリとの転送命令は，1バイト短くなる特別なものが用意されています．これは8080からの遺産であると説明されていますが，実際には固定アドレスの変数よりはローカル変数の使用頻度が高いでしょうから，このような場合には効率の改善にはなりません．

定数値を用いる二項演算

以上の説明では，定数値との間で演算する説明は出てきませんでした．定数値を用いる二項演算の場合，実際に命令コードで示さなければいけないオペランドは1つだけです．ですから，前記のMODフィールドとR/Mフィールドだけが使用され，REGフィールドは命令コードの「拡張」のために利用されます．前記の各二項演算に対して，それぞれ定数演算命令が「別に」存在しています．

これらの命令では，dビットは意味をもちませんが，かわりにsビットとして使用されます．これは，ワード演算（w＝1）のときに，16ビットのデータを引数に取るか，それとも8ビットのデータを符号拡張して使用するかを指定します．ですから，バイト演算のときには命令長は3バイトですが，ワード演算のときでも演算値が－128～127である場合には命令長は3バイトですみます．もちろん，その前に必要ならオフセット値が前置されま

ワンラインNEWS
リコーはパソコンLAN市場の拡大をめざし，メーカーやソフト会社・ディーラーと任意団体「スターラン・ソサエティー」を設立．参加企業は50社前後の見込み．

すので，メモリへの定数の転送は非常に命令語長が長くなります．

AXまたはALと定数との演算の場合にのみ，さらに簡略化された命令が用意されています．これらは命令コードの2バイト目が省かれ，直接バイトまたはワードの引数を取るようになっています．ですから，基本的にはAX／ALを演算に使用するほうが短い命令コードを生成することができます．ただし，定数値が−128〜127であるワード演算の場合だけは，AX以外のレジスタでも変わりません．

なお，定数の転送命令はすべてのレジスタについて簡略命令が存在しています．ワードレジスタへの読み込みは3バイト，バイトレジスタへの読み込みは2バイトです．レジスタに0を設定する場合には，SUB reg，regという命令が使用されることがありますが，ワードレジスタの場合は，このほうが1バイト短くなります．バイトレジスタの場合でも，8086CPUの場合は1クロック実行時間が短くなります．ただし，個人的にはあまりこの方法は好きではありません．

演算系の命令でも，MULやDIVなどは演算対象がAX／DXなどに限定されています．ですから，厳密な意味での二項演算ではなく，単項演算に近いものということができます．単項演算の書式は定数との演算と似ていて，REGフィールドは命令の拡張のために使用されます．これらには，mul／imul／div／idivなどの命令と，シフト・ローテート関係の命令，not／neg命令，inc／dec命令などがあります．

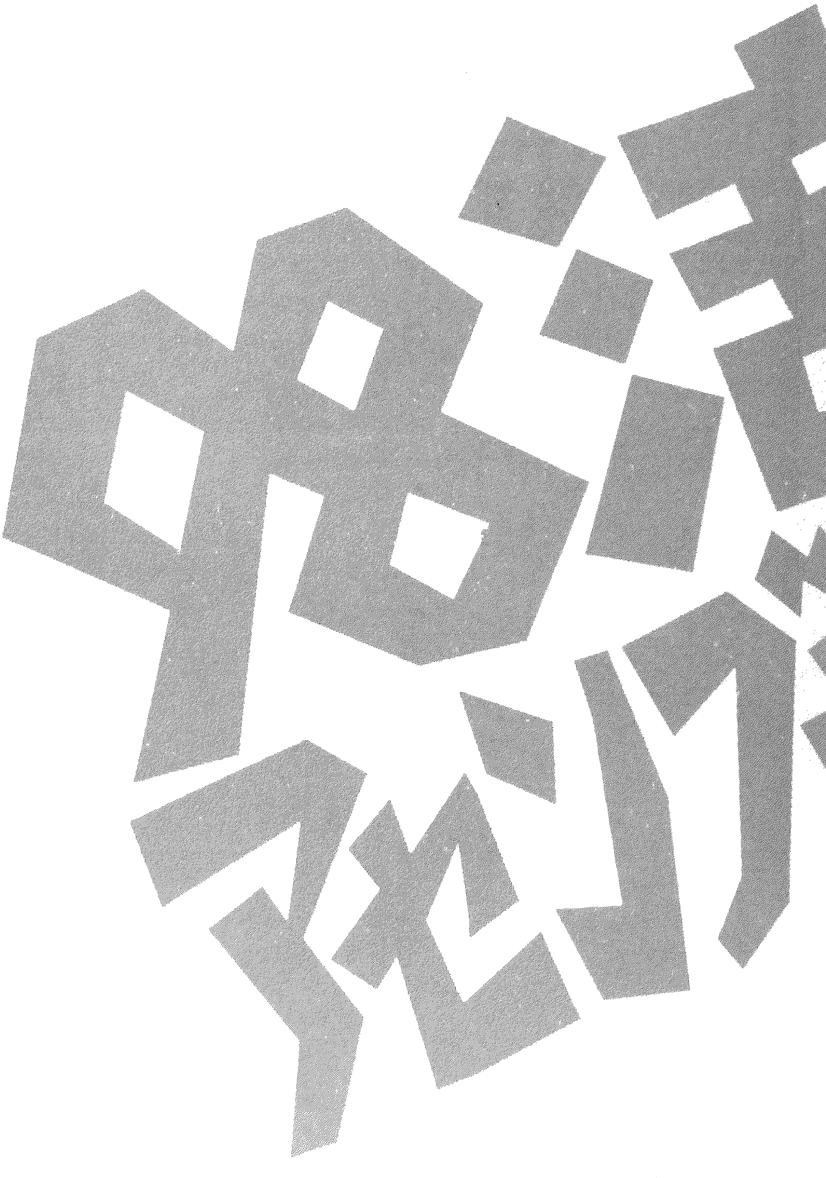
単項演算

単項演算でもinc／decの場合は，16ビットレジスタを使用する場合のみ，1バイトですむ簡略記法が用意されています．ですから，可能ならばバイトインクリメント，デクリメントより，ワード単位のインクリメント，デクリメントを使用するほうがよいということになります．

なお，これらの単項演算に関しては，定数演算のようなAX／ALに対する優遇措置は講じられていません．

汎用レジスタを指定することは，以上のように簡単にできますが，セグメントレジスタを指定する方法は用意されていません．ですから，セグメントレジスタに対する演算（たとえば加算）などはできませんし，そのほかのアクセスに関しても転送系の命令だけに限定されています．

しかも，これらはregフィールドでセグメントレジスタを示すという形を取っていることから，メモリ，レジスタとのやり取りはできますが，定数値を読み込むことはできません．定数値を読み込む命令は，前記のように普通の命令とは異なったフォーマットを採用しており，REGフィールド（この場合は，SREGフィールド）が命令拡張のために使用されているからです．



このように考えると，セグメントレジスタに対する定数値のロードができないことも理解できるでしょう．セグメントレジスタに値を設定するには，メモリ変数を経由するか，レジスタを介して行う必要があるのです．

スタックを利用したpush／pop系の命令は，16ビットレジスタに対してのみ使用が可能です．ですから，これらはセグメントレジスタを使用する場合も含めて，1バイトの命令が割り当てられています．ただし，80386で拡張されたfs／gsについては2バイト命令でサポートされています．そのほかの転送系命令は，sregフィールドが3ビット用意されていたため，問題なくFS／GSに対しての拡張が行えました．

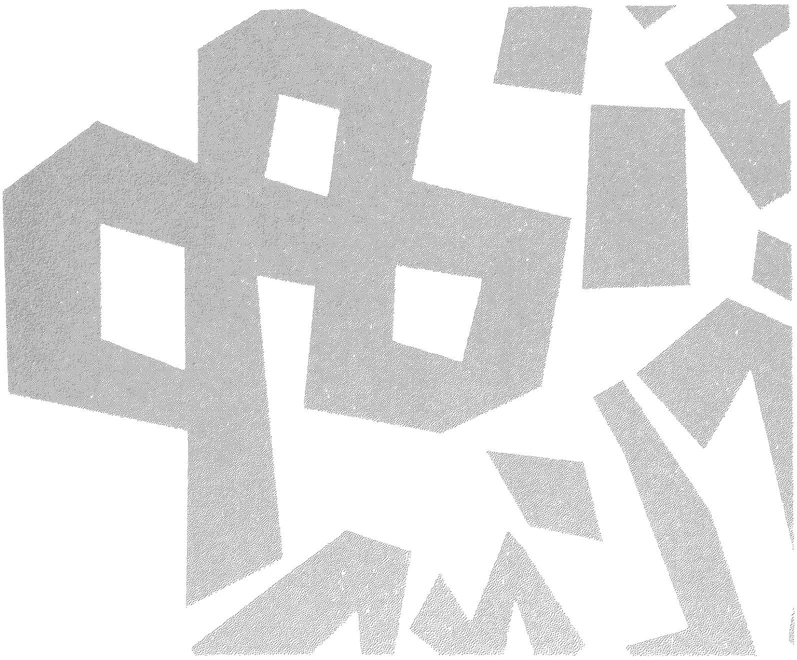
ジャンプ系命令

ジャンプ系命令については，使用頻度が高くなることから命令コードを1バイトに押さえています．条件ジャンプは，全部で16種類用意されていますが，これらは飛び先へのオフセットを1バイトと仮定しています．たいていの場合これでたりるのですが，より遠くに飛ぶ必要がある場合には，前記のように無条件ジャンプと組み合わせる必要があります．条件ジャンプの2バイトオフセットをサポートしなかった理由は，これをサポートすると，さらに16種類の命令コードを消費してしまうということからでしょう．80386では，ワードサイズのオフセットを可能としていますが，この場合の命令コードはコード自体が2バイトになっているため，命令コードサイズは4バイトにもなってしまいます．

無条件ジャンプは，1バイトオフセットと2バイトオフセット用に別々の命令コードを用意しています．さらに，いわゆるFarジャンプで4バイトのアドレスを引数に取る場合も，1バイトの命令コードですませています．

このほかに，無条件ジャンプにはレジスタなどを利用して飛び先を指定する方法があります．たとえば，“jmp [bx]”でbxの示すアドレスに飛ぶことができます．この場合は，単項演算命令と同様なフォーマットを使用して，modフィールドとｒ／ｍフィールドで飛び先アドレスを指定します．

このことから，たとえば“jmp [ax]”というような記法が（80386以上の32ビットアドレッシングを指定しなければ）違法であることは理解できるでしょう．



質の高いソースは命令コードの把握から

さて，以上のように命令コードのビットフィールドまでを考えれば，どのような記法が許されるか，どのようなものが許されないかを比較的容易に判断できると思います．また，生成コードのサイズや実行クロックまで把握しておけば，「ほんとうにクリティカル」なポイントで高い質のコードを書くことができます．これらを理解するためには，CPUチップのマニュアル，あるいはMASMやTASMについてくるリングとじのリファレンスを参照するのがよいでしょう（英語版ではリングとじですが，日本版では平とじ）．

ただし，実際に極限まで最適化する必要があるポイントは多くはありません．たいていの場所では，理解しやすい記述，変更しやすい記述を利用するほうがよいものです．push／pop命令を利用してレジスタを使い回すよりも，メモリ変数，とくにBP経由のローカル変数を利用するほうが便利な場合が多いでしょう．とくに8086では，実効アドレスの計算に時間がかかったので，BP経由のメモリアクセスよりは直接アドレッシングのほうが速かったのですが，80286以上では実効時間に遜色はありません．加えて，多くの場合に命令コードが1バイト短くなるのですから，積極的に利用することを検討するとよいと思います．わざわざ処理系がサービスしてくれるのですから．

ワンラインNEWS
アスキーは7月よりNTTのダイヤルQ2を使ったパソコン通信サービス「アスキーQネット」を開始．パソコンがなくてもオペレータ経由でメールの送受信が可能．

情報処理技術者試験
第2種合格！トレーニング
第6回　商　業

日本工学院専門学校　佐々木秀夫

商業科目の出題範囲

1．商業経済
経済知識(経済用語を含む)，企業活動

2．簿記会計
複式簿記の基本原理，用語，商品売買業の取引と会計処理，工業簿記

3．計算事務
売買・金利の計算方法

4．マーケティング
市場の現状と動向，マーケティングの考え方，マーケティング調査，マーチャンダイジング・プロモーションの意義・用語・考え方，販売価格の決定

5．商品流通，物的流通
流通の歴史・用語・考え方，商品の取引，流通・流通経路の意義・用語・考え方

6．事務管理
事務管理組織，事務分析，事務量算定，文書制度，ファイリング，事務の集中・分散，事務の機械化

7．税務会計
税の概念，法人税

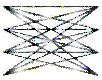
8．経営数学
標本調査の母集団と標本の概念

9．経営情報システム
企業情報（財務・人事）

昭和44年に開始された情報処理技術者試験は，22年たった現在，5種類におよんでいます．出題形式は昭和50年に大幅に改正され整理されました．10年後の昭和60年には出題の範囲を科目の説明として整理して発表しました．

試験の実施を民間に委譲したこと，2種の試験は年2回実施するようになったこと，アセンブリ言語の仕様が変更されたこと，試験の種類が増加した

●図1　取引要素の結合関係



●表1　過去5年（昭和61年～平成3年）の出題問題

年度	科目	内容	出題形式
平成3年春	経営分析 経営分析 簿記会計	売上売上原価の計算，収益性の分析 損益分岐点分析 企業会計原則（一般原則・損益計算書原則）	計算 計算・用語 文章
2年秋	簿記会計 経営分析 簿記会計	勘定科目の区分（P/L，B/S） 安全性分析（流動・負債・当座・固定などの比率） 仮払金の仕訳処理	用語 計算・用語 仕訳・用語
2年春	事務手続 購買業務 簿記会計	手形交換の仕組み（証憑類の流れ） 在庫管理（発注点方式） 取引から決算までの処理	用語 計算・用語 計算
元年秋	簿記会計 簿記会計 販売業務	貸借対照表の区分表示 商品勘定処理と売上利益計算 売上高と売掛債権管理	用語 仕訳・計算・用語 計算・用語
元年春	事務手続 経営分析 簿記会計	購買手続（証憑類の流れ） 収益性・安全性の分析（資本・負債・流動） 企業会計原則（一般原則）	用語 計算・用語 文章
昭和63年秋	販売管理 簿記会計 簿記会計	売上数量の損益分岐点 経過勘定（費用の繰延） 貸借対照表の区分表示	計算 計算・用語 用語
昭和63年春	経営形態 簿記会計 販売計画	株式・有限などの法律定義 損益計算書の区分表示 売上商品の目標利益	文章 文章・用語 計算
62年秋	マーケティング 工業簿記 簿記会計	価格政策（慣習・均一・特価・端数・見切・再販売） 原価の構成要素 試算表	文章・用語 計算・用語 仕訳・計算・用語
62年春	簿記会計 計算事務 簿記会計	貸倒損失（差額補充法） 利回り（利付債券・割引債券） 棚卸商品の評価（低価法）	仕訳・計算・用語 用語 計算
61年秋	流通・物流 計算事務 販売管理	流通経路と取引方法 定期預金の金利計算 商品別売上利益計算	用語 計算 計算
61年春	マーケティング 簿記会計	流通経路と販売政策 本支店取引の仕訳	文章・用語 仕訳と計算

ことなど，世代に応じて，調査・研究・変更をくりかえしています．昨年新たに民間人を主体とする研究会が発足して，近い将来，大型汎用システムからワークステーションとネットワークの時代に対応させるために新しい内容に変更するための整備が進んでいるようです．

今月号は商業科目の出題範囲と実際に出題されている問題を比較して，その傾向について調べ，受験者に大いに参考になるように整理してみました．

昭和60年頃までは簿記・会計・商品売買の粗利計算などの出題がおもなものでしたが，昭和60年から63年頃の間に，商業経済・流通・税金・経営組織・マーケティングなどの問題が連続して出題されました．平成になって，購買手続や手形交換の仕組（金融）などの例外と思われるような出題もありますが，最近の2か年間では経営分析・企業会計原則・簿記・会計などと，昭和60年頃の出題傾向に近くなっています．もっとも，企業会計原則は新しい出題です．

現在までに，システム設計に必要な技術の事務管理，経営管理に必要な技術の経営数学，経営情報システムなどに直接関係する問題はまったく出題されていません．そこで，近年の傾向から出題頻度の高い科目を並べてみますと，

1．簿記会計
2．財務諸表（P/L）（B/S）
3．経営分析（財務分析）
4．商品売買の計算（目標利益）
5．その他

の項目に整理できます．選択問題は3題出されるので，以上の科目の出題ポイントを要領よく勉強しておけば2題は確実に解答できます．

簿記会計

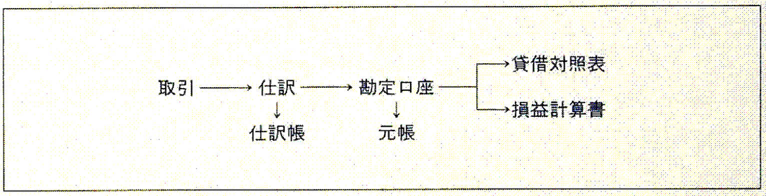
技術者試験の目的は，会計情報の認識と会計原則に従った正しい会計処理と経営管理の理解などにあり，日商簿記検定の2～3級の範囲と考えてよいと思いますが，検定試験は記帳や計算を目的として帳票類の作成と正しい計算に主眼が置かれていることに注意してください．

取引の認識・取引の要素・仕訳・試算表・精算表・損益計算書・貸借対照表などの意義と機能，作成手順など，仕入・売上・繰越・棚卸などを要素として売上利益を計算する商品勘定，繰延・見越といった経過勘定，評価性・負債性の引当金勘定，減価償却の意義と基本手続，継続記録法による棚卸資産の評価，仮払金の精算，損益計算書・貸借対照表の表示区分と代表的な勘定科目などは確実に理解しておかなければなりません．

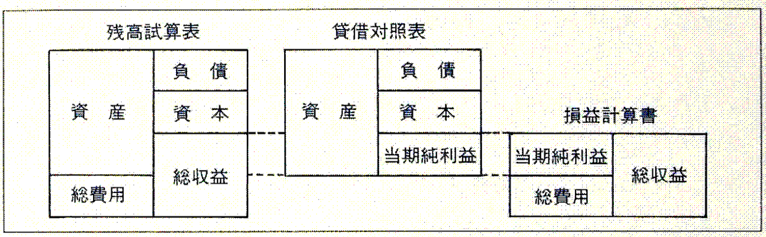
財務諸表（P/L，B/S）

簿記論，商法計算書類規則，企業会計原則，財務諸表等規則，法人税法などで必ず作成しなければならない財務諸表は，貸借対種表（B/S）と損益計

●図 2　記帳の流れ



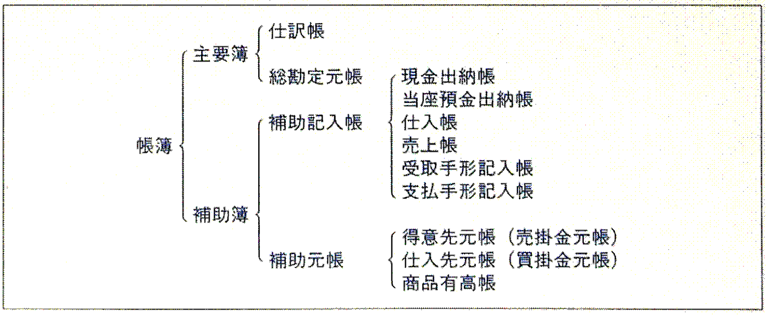
●図 3　試算表と財務諸表の構造図



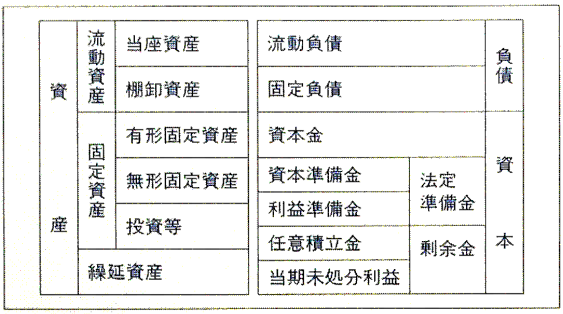
●図 4　経過勘定

費用の繰延……費用から控除して資産へ加える（前払）
収益の繰延……収益から控除して負債へ加える（前受）
費用の見越……費用へ加算して負債へ加える（未払）
収益の見越……収益へ加算して資産へ加える（未収）

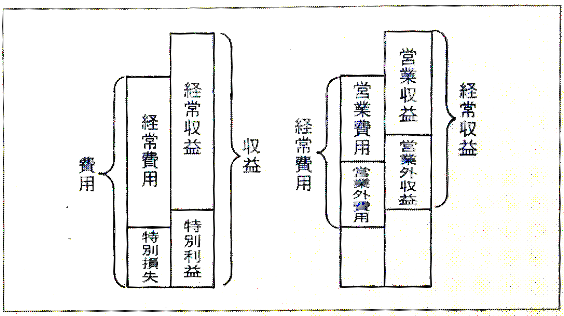
●図 5　主要簿と補助簿



●図 6　財産表示区分



●図 7　損益表示区分



●図 8　財産法と損益法

［財産法］　期末資本－期首資本＝純利益（純損失）
↓
同額
↑
［損益法］　総収益　－　総費用＝純利益（純損失）

●表 2　貸借対照表勘定科目の大分類と表示方法

資産	流動資産	当座資産	現金，預金，有価証券，受取手形，売掛金，前払金，支払手付金，貸付金，仮払金，未収金，立替金
		棚卸資産	商品（仕掛品，半製品，貯蔵品）
	固定資産	有形	備品，土地建物，機械，船舶，車両，運搬具，建設仮勘定
		無形	特許権，実用新案権，意匠権，営業権，商標権
		投資その他	投資有価証券，子会社株式，長期貸付金，減債基金
	繰延資産		創立費，開業費，開発費，建設利息，新株発行費，社債発行費

負債	流動	支払手形，買掛金，前受金，受取手付金，手形借入金，当座借越，仮受金，未払金，預り金前受収益
	固定	社債，長期借入金，退職給与引当金
資本	資本金	資本金
	法定準備金	資本準備金（株式払込剰余金，減資差益，合併差益）利益準備金
	剰余金	任意積立金，未処分利益剰余金

●表 3　損益計算書勘定科目の大分類と表示方法

費用	営業費用	仕入（売上原価），通信費，保険料，営業費，広告費，発送費，貸倒損失，消耗品費，支払家賃，租税公課，減価償却費，光熱費
	営業外費用	支払利息，支払割引料，売上割引，社債利息，有価証券利息，雑損
	特別損失	固定資産売却損，除却損

収益	営業収益	売上，割賦売上，未着売買益，積送品売買益
	営業外収益	受取利息，受取割引料，仕入割引，有価証券売却益，受取手数料雑益
	特別利益	貸倒引当金戻入，償却債権取立益，固定資産売却益

Ⅰ．一般原則

① 真実性の原則

企業会計は，企業の財政状態および経営成績に関して，真実な報告を提供するものでなければならない．

② 正規の簿記の原則

企業会計はすべての取引につき，正規の簿記の原則に従って，正確な会計帳簿を作成しなければならない．

③ 資本取引・損益取引区分の原則

資本取引と損益取引とを明瞭に区別し，特に資本取引から生じる資本剰余金と，損益取引から生じる利益剰余金とを混同してはならない．

④ 明瞭性の原則

企業会計は，財務諸表によって利害関係者に対して必要な会計事実を明瞭に表示し，企業の状況に関する判断を誤らせないようにしなければならない．

⑤　継続性の原則
企業会計は，その処理の原則および手続きを毎期継続して適用し，みだりにこれを変更してはならない．
⑥　保守主義の原則
企業の財政に不利な影響を及ぼす可能性がある場合には，これに備えて適当かつ健全な会計処理をしなければならない．
⑦　単一性の原則
株主総会に提出するため，租税目的など，種々の目的が異なる形式の財務諸表を作成する必要がある場合，それらの内容は信頼し得る会計記録に基づいて作成するものであるから，政策の考慮のために真実の表示をゆがめてはならない．
2．損益計算書原則
①　費用収益対応の原則
損益計算書は，企業の経営成績を明らかにするため，一会計期間に発生したすべての収益とそれに対応するすべての費用とを記載し，当期純利益を表示しなければならない．
②　総額主義の原則
費用及び収益は，総額によって記載することを原則とし，費用の項目と収益の項目とを直接に相殺することによってそのすべてまたは一部を損益計算書から除去してはならない．
③　発生主義の原則
すべての費用および収益は，その支出および収入に基づいて計上し，その発生した期間に正しく割り当てられるように処理しなければならない．ただし未実現収益は，原則として当期の損益計算に計上しなければならない．
3．貸借対照表原則
①　完全性の原則
貸借対照表は企業の財政状態を明らかにするため，貸借対照表記入日におけるすべての資産，負債及び資本を記載し，株主や債権者，その他の利害関係者にこれを正しく表示するものでなければならない．
②　明瞭性の原則（区分計算表示の原則）
資産・負債及び資本は，適当な区分，配列，分類及び評価の基準にしたがって記載していかなければならない．
③　総額主義の原則
資産・負債及び資本は総額によって記載することを原則とし，資産の項目と負債または資本の項目とを相殺することによって，その全部または一部を貸借対照表から除去してはならない．

算書（P/L）の2つです．それぞれについて大分類と表示区分，形式と配列方法などは憶えておかなければなりません．

企業会計原則は一般原則・損益計算書原則・貸借対照表原則の3つから構成されています．それぞれの内容を理解しておく必要があり，とくに，1種との関連が深い損益計算書と貸借対照表の原則は注意が必要です．

損益計算の区分構造

売上高－売上原価＝売上総利益
売上総利益－販売費および一般管理費
　＝営業利益
営業利益＋営業外収益－営業外費用
　＝経常利益
経常利益＋特別利益－特別損失
　＝税引前当期利益
税引前当期利益－法人税等引当金
　＝当期利益
当期利益＋前期繰越利益
　＝当期未処分利益

経営分析（財務分析）

経営分析の種類には，取引先の信用度，投資先の経営状態，自社の経営管理などがあります．方法には実数と比率を使います．安全性・収益性・生産性の3つについて代表的な指標を示します．

■安全性の分析

$$当座比率=\frac{当座資産}{流動負債}\times 100(\%)$$

流動負債の支払に充てる当座資産がどれだけあるかを示す比率です．

$$流動比率=\frac{流動資産}{流動負債}\times 100(\%)$$

流動負債の支払に充てる当座資産がどれだけあるかを示す比率です．

$$固定比率=\frac{固定資産}{自己資本}\times 100(\%)$$

固定資産の運用に自己資本をどれだけ充てているかを示す比率です．

$$資本負債比率=\frac{自己資本}{負債}\times 100(\%)$$

自己資本と負債（他人資本）の割合を示し，長期的な返済能力を示しています．

■収益性の分析

$$資本利益率=\frac{利\quad益}{資\quad本}\times 100(\%)$$

企業活動に使われている資本（当期平均）に対する利益の割合を示すもので，高いほどよいのです．

$$売上利益率=\frac{利\quad益}{売上高}\times 100(\%)$$

売上高に対する利益の割合です．利益率の計算にはいろいろあります．

$$資本回転率=\frac{売上高}{資\quad本}\ （回／年）$$

1年間に資本が何回転したかを示します．

■生産性の分析

$$労働生産性=\frac{付加価値}{従業員数}$$

従業員１人あたりの付加価値額を示しています．

$$労働装備率=\frac{有形固定資産}{従業員数}\times 100(\%)$$

従業員１人あたりどのくらいの設備で装備されているかを示しています．

$$設備利用度=\frac{売上高}{有形固定資産}\times 100(\%)$$

有形固定資産の何倍の売上を実現したかを示しています．

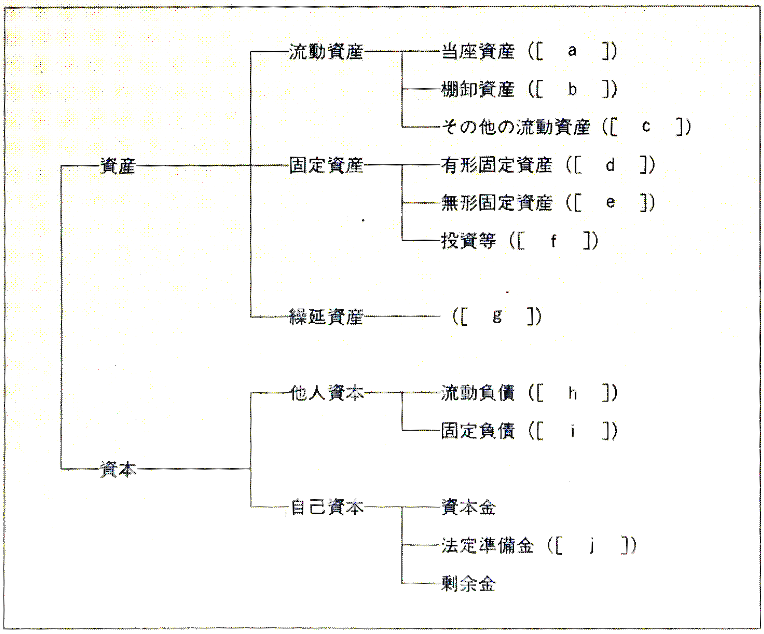
■損益分岐点分析

$$損益分岐点売上=\frac{固定費}{1-\dfrac{変動費}{売上高}}$$

EXAMPLE 1

会社の資産及び資本の区分を示す次の体系図中の［　］に入れるべき適当な字句を解答群の中から選べ．

●資産と資本の区分



<解答群>

ア　売掛金	イ　買掛金
ウ　開業費	エ　建設仮勘定
オ　子会社株式	カ　仕掛品
キ　退職給与引当金	ク　特許権
ケ　前払金	コ　利益準備金

解　説

貸借対照表の表示区分に関係し，各勘定科目がどの区分に属しているかを問う問題です．資産も資本も流動性配列法によって表示されており，解答に示されているそれぞれの勘定科目がどの区分に属しているか，勘定科目の性質をよく理解しておかなければなりません．

解　答

a－ア，b－カ，c－ケ，d－エ，e－ク，f－オ，g－ウ，h－イ，i－キ，j－コ

EXAMPLE 2

「企業会計原則」における「損益計算書原則」に関する次の記述中の［　］に入れるべき適当な字句を解答群の中から選べ．

(1)［　a　］は，当期純利益に前月繰越利益，一定の目的のために設定した積立金のその目的に従った取崩額，中間配当額，中間配当に伴う利益準備金の積立額等を加減して表示する．

(2)［　b　］は，売上総利益から販売費及び一般管理費を控除して表示する．

(3)［　c　］は，経常利益に特別利益を加え，これから特別損失を控除して表示する．

(4)［　d　］は，売上高から売上原価を控除して表示する．

(5)［　e　］は，営業利益に営業外収益を加え，これから営業外費用を控除して表示する．

<解答群>

ア　売上総利益	イ　営業外収益
ウ　営業利益	エ　経常利益
オ　限界利益	
カ　税引前当期純利益	
キ　当期純利益	
ク　当期未処分利益	ケ　特別利益
コ　内部利益	

解　説

損益計算の区分構造を示し，その計算手順を問う問題です．財務諸表の表示方法には勘定式と報告式があり，一般に報告式の方が多いようです．売上総利益から当期未処分利益までの計算手順は原則によって定められていますので，その手順について確実に理解しておかなければなりません．

解　答

a－ク，b－ウ，c－カ
d－ア，e－エ

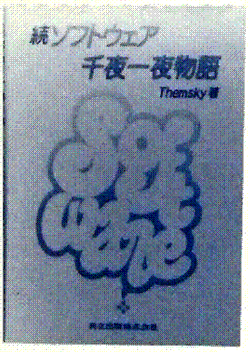
Book Review

■中村正三郎

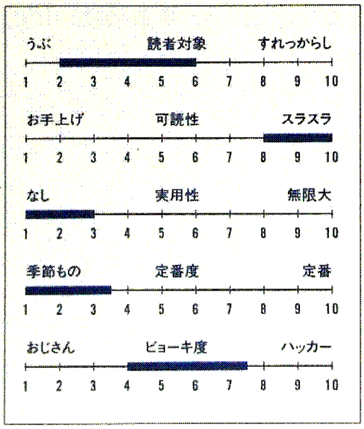
アカデミックフレーバーは パソコン関係の人たちに新鮮

■続ソフトウェア 千夜一夜物語

Themsky 著
共立出版　2580円



■評価チャート



いきなり続編のほうを紹介してしまう．本書は，「ソフトウェア千夜一夜」シリーズの第2弾なのだ．このシリーズ，『bit』誌に連載されたものをまとめたアンソロジーで，コンピュータにまつわる軽い読物集である．匿名の7人によって綴られる話題は多岐にわたっている．

著者たちは，大学関係者や研究者といったアカデミック方面の人であり，本誌の読者層であろうパソコン関係の人たちとは，違う計算機環境に住んでいる．当然のように，内容は，ちょっとアカデミックフレーバーがする．

このアカデミックフレーバー，パソコンの世界しか知らない人には新鮮に映るだろう．逆にそこから発する啓蒙臭が気になるかもしれないし，その限界すら見えてしまうかもしれない．

これは掲載誌である『bit』誌が，アカデミック世界のサロン誌の性格が強い雑誌であることも影響している．

前作は，「たかがなんとか，されどかんとか……」など，似たものを集めて数部に分けていたが，今回は千夜一夜物語風に前夜を受けてつないでいき，最後まで流れを作るように構成してある．しかし，自然な流れというには少々強引である．あっち飛びこっち飛びの印象が強くなってしまった．これはこれで面白いのだが，評者は前作のほうが，まとまりのよさからくるパワーを感じた．今回は散漫な印象を受けてしまうのである．

前作の「データ型，その継承論」「証明を型にはめると」のように，理論的で読み流すにはハードな話題が少ないことも，パワーを感じなかった原因だろうか．今回，理論的な話題は，唯一最後の「第29夜 止れ！」だけである．その分，読みやすいともいえるのだが．

今，最後が第29夜と書いたので，勘のいい読者は，29編の作品が収録してあることに気づいただろう．残念，大間違いなのだ．本書に収められているのは14編であり，前作が15編集めてあったので，今回は第16夜から始まって第29夜までなのだった．

評者が読んで面白かったのは，「第19夜 並列計算に関する99の誤解」と，「第24夜 たかがシンタクス，されど辛多苦す」以降，「第25夜 間違い探し」のプログラミング言語モノだった．第24夜以降は，評者自身大変うまく流れに乗っていけて，すいすい読めた．評者の理解がおよぶ範囲の話題が続くせいだ．ところが，第29夜で，プログラムの停止性問題が出て，寝苦しい夜になってしまった．:-)

こういう話，評者は学生時代はなんとかなったものの，もはや全然頭がついていかない．先に述べた前作の2編は，読んだとき，脳のしわがかゆくなった．負け惜しみで書くが，このような理論面での啓蒙的話は，本書にももう1つくらい収めてほしかった．脳のしわがかゆくなるのは，脳の老化を防ぐためにはいいことなのだ（←だと思う）．

本シリーズの特徴といっていいだろうが，索引を眺めていると，普通では見出しに取りあげないようなミョーな言葉がある．たとえば，本書でいえば，「アバウトな感覚」「覚えておいてからフェイルする」「スーパーペンギン」「バンバン・スタイル」などなど．前作で

ワンラインNEWS
オムロンはソフトウェアの開発や事務作業などを一括処理できるワークステーションを結ぶシステムの外販を開始し，このソフト専用の営業部隊を新設した．

もミョーなのがある.「悪名高きマイクロアセンブラ」「中途半端な壊れ方」「Prologがきらい」「Prologが好き」などなど．こういうところにも，このシリーズのユーモア感覚が現われている.

本書をお読みになるなら，是非，前作もどうぞ．またこのシリーズの前身として『bit』に連載されていた「プログラミング・セミナー」も共立出版から本になって出ている．興味のある方はご一読を.

今いちばん活気があるのはネットワーク関連書

古田島義和（神田・書泉グランデ）

日本でもようやくネットワーク環境が整いはじめ，普及率も目を見張るものがあります．今回は，その中でもとくに注目を集めているUNIX Networkについてと，UNIX関連図書の数々を紹介したいと思います．まず，TCP/IPという言葉は，プログラマの大半はすでにご存知でしょう．'89年にソフトリサーチセンターから「TCP/IP」が出版されて以来，オーム社「異機種接続とTCP/IP絵とき読本」，共立出版bit別冊「TCP/IPによるネットワーク構築」（これは，Prentice Hall（以下PH）「Internetworking with TCP/IP」Comer著の訳本）と続々出版され，ロングセラーになっています.

このTCP/IPは，バークレー版UNIX（BSD）の拡張機能を備えた4.2BSDから実装され，4.3BSDで確固たるものを築きあげました．最近の図書ではPH「The Simple Book:An Introduction to Management of TCP/IP」（Luy Rose著）が最もよく売れている1冊です．最新刊として上記でもあげたPHの「Internetworking with TCP/IPII」（Comer著）とAddison Wesley（以下AW）の「TCP/IP and NFS」（Santifaller著）もほぼ同時期に出版され，予想通りの売れ行きを見せています．いうまでもなくNFSは，「Network File System」のことで，SUN Microsystems社の共同設立者でもあるBill Joy氏が設計したシステムです．ほかにO'Reilly（以下OR）の「Managing UUCP and c/senet」の改訂新版に続き，同社の「Using UUCP and c/senet」も改訂新版になり，よく売れています．ネットワークは，今いちばん活気のある分野のようです.

次にUNIXセキュリティに関する図書が出版されました．ORの「Practical UNIX Security」がそれです．前にAWの「UNIX System Security」が売れましたが，今回はそれを上回ることまちがいなしです．すでに訳本も出版されています．Howard W. Sams社の「UNIX System Security, Revised Edition」は，HBJ出版から「UNIX System Vシステムセキュリティ」として出版．システムセキュリティの一般的な解説書ですが，よくまとめられていておすすめの1冊です.

以前紹介したAWの「Design & Implementation of 4.3 BSD UNIX Operation System」（Leffler著）の訳本が丸善から「UNIX 4.3 BSDの設計と実装」として出版．予想をはるかに超える売れ行きにビックリ．来月のベスト1位は間違いないでしょう．同じころに洋書の方では，同じタイトルの「Answer Book」が入荷して爆発的な売れ方をしています．和書のベスト10，第1位にはパソコン通信が入っており，こちらも楽しみな分野になるでしょう．第2位の工業調査会の「人工現実感の世界」も健闘しています．この類の分野が低迷しはじめて早2年ほどが過ぎ，不安がありましたが，ホットした感じです．洋書ではAWの「Artifical RealityII」（H. W. Krueger著）が入荷してから動きもよく期待の1冊です.

■和書&洋書ベスト10

（書泉グランデ調べ：1991年6月1日～6月30日）

1	WTERMで今日から始めるパソコン通信	秀和システム
2	人工現実感の世界	工業調査会
3	C言語による最新アルゴリズム事典	技術評論社
4	UNIX日記	ソフトバンク
5	THINK Cプログラミング講座	技術評論社
6	SUNシステム管理	アスキー
7	日本語入力読本	翔泳社
8	安全太郎の夜	河出書房新社
9	SUNシステム管理	アスキー
10	入門C言語　改訂新版	アスキー

1	Programming Perl	O'Reilly
2	ResEdit Reference 2/E	Addison Wesley
3	Compilers	Addison Wesley
4	Undocumented DOS	Addison Wesley
5	Programming Windows 2/E	Microsoft Press
6	Object-Oriented Analysis 2/E	Prentice Hall
7	MH & zmh	O'Reilly
7	Object-Oriented Modeling and Design	Prentice Hall
7	Mathematica Jorunal V1 Issue 2 Fall '90	Addison Wesley
10	System Performance Tuning	Prentice Hall

◆ ◆ 佐々木哲のニューヨーク ◆ ◆

N.Y. システム通信

SYSTEM N Y NEWS

Recessionではなく Restructure

景気後退ということが，アメリカでよく聞かれるようになってしばらくたちます．ソフトウェア，ハードウェアのコンピュータ産業では，景気後退というより，業界の構造が変わりつつあるようです．

それにしても，7月はじめの週は大騒ぎでした．Microsoftに対抗するIBM，Appleの提携の驚きがさめないうちに，BorlandによるAshton-Tateの吸収合併発表です．この分野には，名門などありません．その余裕がない．この波が日本にこようとしている今，気を抜かず前を見ているか，根気よく正しいことを続けていくことが要求されています．日常に流されて，こんなことやっていていいのかな？　と思っているならすぐにやめましょう．

地元で気楽なPC Expo

6月末，地元ニューヨークで開かれたPC Expoに行ってきました．おもに企業バイヤー向けのPCのショーです．規模からいえば，5月末のComdex Springと肩を並べるほどで，かえってこちらのほうが興味深いものが見れます．

やはり，地元で開かれるショーというのは勝手がわかって気楽でいいですね．狭いマンハッタンですから，その気になれば事務所から歩いて行ける．ハドソン川沿いのJacobs Jarvics Centerという，巨大な温室のような建物で開かれます．ショー会場への交通はタクシーが一番便利．あるいは34番のバスが便利でしょう．帰りは，2，3ブロック歩いて8th Ave.あたりまで出てタクシーをつかまえましょう．Comdexと違って主催者の提供するシャトルバスは当てにならない．

もっとも，今年は何も新しいものがないショーでしたが，それでも実際にものに触れて，メーカーの人間と話せるのが何よりのよさ．ラスベガスという行楽地で開かれる秋のComdexほど派手じゃないけど，アトラクションもおもしろいです．

各社の486の50MHzマシン，NCR 3125 pen-based computer，IBMのOS/2 2.0，Lotus 1-2-3 for Windows．各種マルチメディア用デバイス，とくに，Sound Blasterのような音源がめだちますね．

台湾の進出，生産基地化

それにつけても，ショーに行くたびに彼らのパワーには圧倒されます．台湾だけでなく，シンガポールなど東南アジア各国．いまでは，台湾は全世界の60%のパソコンを生産しているというデータもあります．パソコン分野での新技術が種切れになるにつれ，またパーツ化，分業化が進むにつれ，開発力のある大企業のメリットが薄れてきています．誰が作っても同じものになってしまう．ましてや，すべての根元をINTELとMicrosoftに押えられていてはなおさらです．あるいは，一歩進んだものを作っても，すぐに追いつかれてしまいます．たとえば，VGA，386SXのNotebookコンピュータ．Compaqや東芝が切り開いた分野ですが，1年もたたないうちに安価な各種Notebookがあふれている始末です．台湾では，国家プロジェクトとして共同で開発，生産をすすめているとか．少しずつデザインが違うNotebookが，さまざまなブランドで出てきています．

いままで，台湾からでは米国への販売が1つネックでした．いまでは，中小のところでもカリフォルニア州に輸入基地というべきものを設定しています．アメリカ国内のディーラーは台湾からではなく，カルフォルニア州から“輸入”する．アメリカのパソコンIBM互換機を知り，マーケットを知っていてこそできることです．

Tsengの新VGAチップ

このコラムでも何回か触れている最高速のVGAチップセット，Tseng[*1] ET4000は，新たに1280×1024×16色のモードを加えて高機能化をはかっています．さらに，バスを通さずCPUに直結できるインターフェースを用意するとか，マザーボード上に直接VGAが載る形になります．

さらに，この秋にはET4000XXとして，現在のものとピン互換でありながら，GUI向けの描画機能を持ったチップセットを出荷する予定です．ATIの8514/Ultra

ワンラインNEWS
アスキーは，ワープロ専用機で作成した文書・図形データをパソコンで利用したり，データ変換を可能にするワープロ「New ASKeyword」を発売した．

と同様，専用のドライバにより，MS-Windows，AutoCADなどが高速に実行できるものです．先のCPUとの専用バスと組み合わせれば，最高速のWindowsマシンができあがるわけです．

既存のET4000のユーザーにも救いがあります．この8月には，いままでより3～5倍速い新しいWindowsドライバがリリースされる予定だとか．Orchid，STBなどのVGAボードを使っているユーザーは，このドライバの入手を待つだけです．Tsengは，ATI，Tridentなどと違ってBBSを持ちませんが，各メーカーからサポートされ次第，CompuserveのWindows関係のフォーラムで手にはいるようになるでしょう．

ZmodemをTurbo Pascalで

Turbo Pascal使いには，昔からの強い味方がTurbo Power Softwareです．Turbo Professinal，Object Professionalなどの強力なプログラムライブラリを，すべてソースコード付きで安価に提供してきてくれました．彼らからまた新しいライブラリ*2が発売です．今回は通信用のライブラリです．またもいたれりつくせりのものです．特筆すべきは，ファイル転送プロトコルとしてZmodemのサポート，さらにはデータの圧縮展開のサポートです．

ところで，ずっとBorlandのお膝元にいたTurbo Power Softwareですが，今度，よりよい開発環境を求めてコロラドへ引っ越したとか．

なお，ZmodemはC言語のライブラリでは，Greenleaf Comm Library*3，Solid Software*4のSolid Linkでサポートされています．後者はModula-2のバージョンもあります．

データ圧縮チップStacker

基礎技術を持つと強いものです．そのひとつがデータ圧縮．Stac Electronicsは独自の高速圧縮技術を開発し，チップにまとめました．昨年秋に，それに基づいた製品を発売しています．ディスクの容量を倍増するユーティリティ，Stacker*5です．

製品は2種類あります．ISAバスのボードが付いてくるものとソフトウェアのみのものです．ハードウェア版では，圧縮，展開がボード上のチップにより行われますので，高速しかも圧縮率もあがります．また，デバイスドライバのサイズも小さくなります．

インストールすると，ディスクの内容を圧縮しながら指定された容量の仮想ディスクを既存のハードディスク上に作ります．たとえば，HD 1台のシステムに20MBのディスクを指定してインストールすると，10MBのhidden fileが作られます．これがデバイスドライバにより，Dドライブとしてアクセスされます．このインストールはかなり賢く，config.sysの内容をチェックしながら，まだ仮想ディスクにアクセスできないブートアップ時に必要なファイルだけを残して，後はコピーして圧縮してくれます．

後は普通のドライブとして使えます．ユーザーは圧縮されていることを気にする心配はありません．ファイルの種類により，圧縮の程度の差はありますが，だいたい2：1から1.5：1の圧縮比が得られるようです．

気になるスピードの低下もあまりありません．とくに，ハードウェア版を使った場合，アクセスの遅いディスクではデータ転送量が減るため，かえって高速化するケースもあるとか．ソフトウェア版も比較的高速なCPUのパソコンでは気になりません．

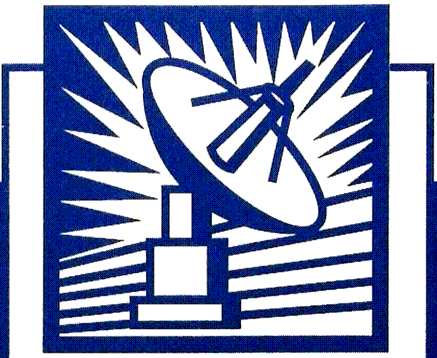
互換性に付いては，MS-Windows 3.0の386 Enhanced modeを含めて使用可能です．ただ，使ってみた限りでは，高速で通信している場合や一部のFEPなどと相性が悪いようです．そのような場合は，ディスクの一部を圧縮せずに残しておいて，そちらを使うことになります．とくに，容易にディスクのupgradeができないNotebookパソコンのユーザーなどにおすすめです．

この圧縮チップは，本来ならばディスクやパソコンにもともと装備されているのが望ましいですね．ユーザーからまったく見えない形で装備されている．SCSIなどのディスクかコントローラに載っているような形です．

＊1 Tseng Labs，phone (215) 968-0502
＊2 Async Professional $139 with source code
Turbo Power Software PO Box 49009，Colorado Springs，CO 80949-9009，phone (719) 260-6641
＊3 The Greenleaf COMM Library for C $359 with source code
The Greenleaf COMM++ Library for C++ $199 with source code
16479 Dallas Parkway #570，Dallas，TX 75248，phone (214) 248-2561
＊4 Solid Link with Zmodem
for C or Modula-2 with source code $498
for C++ with source code $547
Solid Software
＊5 Stacker
Software version $149，Hardware version $229
Stac Electronics Inc.，5993 Avenida Encinas，Carlsbad，CA 92008，phone (619) 431-7474

SYSTEM N Y NEWS

Illustration : NOBUKI FUKUHARA



西川和久の

希望的観測

驚異の低価格!!
アキバで486/33MHzマシンが25万円!?

1 激安マシン発見!! RAM4MBにSCSIまでつけて……

最近，DOS/Vにつられてかなり安いマシンの広告が目につくようになった．もちろん海外から個人輸入すれば，それなりに安いのだが，交渉はすべて英語だし，トラブルやサポートの面など，いろいろ不安要素があるため，あまり一般的ではない．やはり，国内で買えるほうがいいに決まっている．雑誌や秋葉原を物色していて，国内で買える激安のマシンをついに発見した．

秋葉原にあるコムサテライト（株）では，ユーザーの希望に応じて各ユニットを組み合わせて，マシンを組み立ててくれるのだ．私事で恐縮だが，会社の開発マシンはひと昔前のものが多く，最速マシンでi80386/20MHz，平均的に見てi80386/16MHz程度で，パワー不足を感じていた矢先なので，早速見積っていただくことにした．

「会社ではNetwareがあるのでHDは必要ないし，今使っているマシンのリプレースなので，モニターもVGAもキーボードも必要ない．『i80486/33MHz Cache 64KB，RAM 4MB，3.5'inch FD×1，SCSI I/O』この程度でAboutどれくらいになりますか??」

と聞いたところ，帰ってきた返事にビックリ．「25万円程度かな!!」．うーん，これは激安だ．いろいろ話を聞いたところ，各ユニットのブランドやコンフィグレーションにもよるが，モニターや200MBのHDを加えても50万円程度だそうだ．ひさびさに無条件でビックリした．この値段ならリスクをしょって，海外から個人輸入する必要もない．即，発注してしまった．実際に運営して問題がなければ，すべてのマシンを入れ換えてしまおうとさえも思っている……．(^^;

気になるサポートの点でも面白い返事があった．「店がある限り永久サポート」だそうだ．世の中まだまだ楽しいお店があるようです!! :-)

2 腐っても鯛!? ノーブランドでも486!?

もちろん，このマシンは有名メーカーのブランド商品ではないので，いろいろ不安になる人も多いと思う．また，IBM PC互換機でDOS/Vを使うと日本語環境になるといっても，キーボードが日本語対応していないため，PS/55用のキーボードを付けるか，SimkeyやSimkeyAXなど，オンラインソフトを使ってちょっと小細工する必要はある．現在，DOS/V対応のアプリケーションは少ないため不自由なこともあるが，プログラミングを主体に考えた場合，米国最新版の開発環境が使え，MifesやVzが動き，通信ソフト，一太郎，Lotus 1-2-3，ATOK7，VJE-βがあればたいした問題ではない．DOS/V自体が，いろいろなところで注目されている関係上，今後アプリケーションも増えるであろう．

同等なスペックで有名メーカーの国産マシンを購入すると，5～10倍ほど価格の開きがあるのだ．“安心”

ワンラインNEWS
富士通は米エクサバイト社が開発した8ミリビデオテープによるコンピュータ用記憶装置「EXB8200」を同社のワークステーションに採用することを決めた．

とか“累計で対応アプリケーションが……”という言葉に対して，これだけの価格差はいったいどういう意味をもつのだろうか!?　少なくとも私にとっては，何の意味もない．

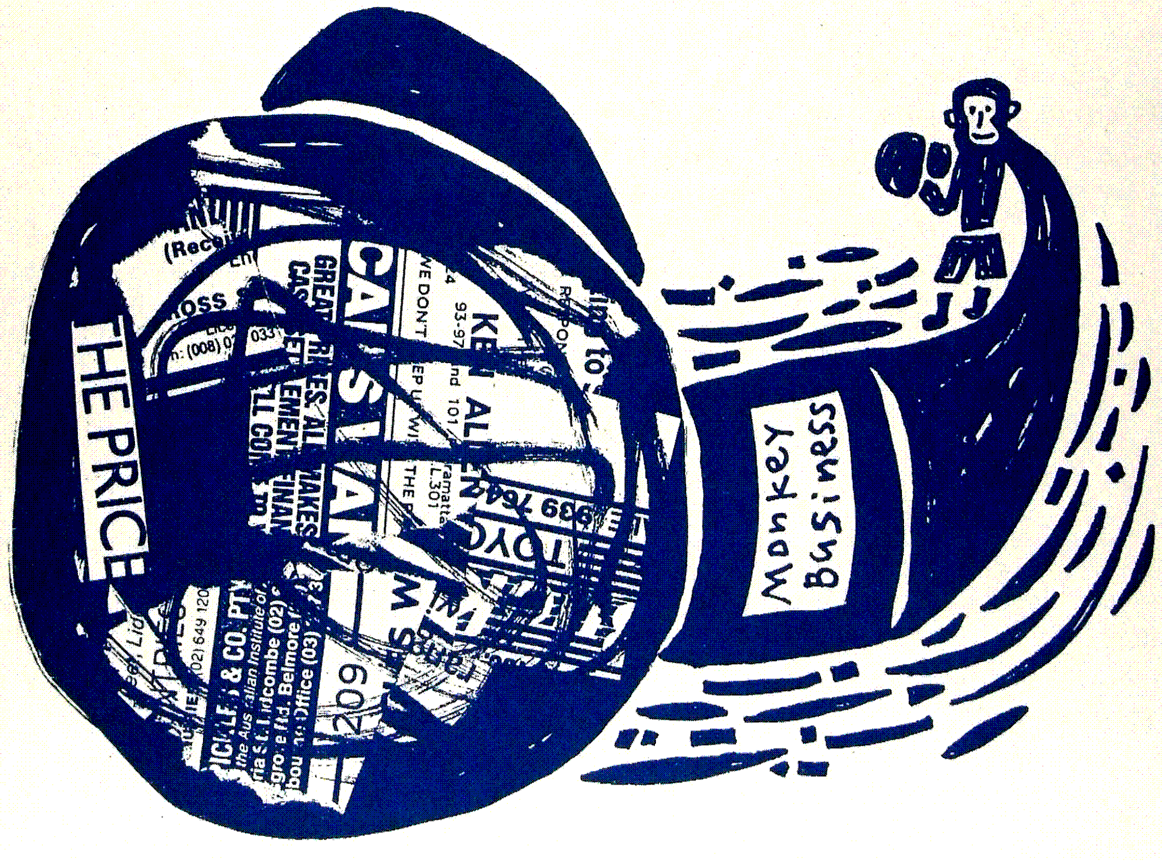
ノーブランドでも，i80486/33MHzはi80486/33MHzである．DOS 5.0＋DOS/Vの日本語環境下（8月号参照）でNetwareに問題なく接続され，日本語Windowsはビシバシ動き，重たいMicrosoft-CもまるでTurbo-Cのように動く．DOS/Vのスクロールは，一般的に遅いといわれているが，ビュイーンっとブッ飛んで行く……．快適の一言である．「腐っても鯛」とはよくいったものだ．

3 1ドル＝140円とまではいわないが……マシンも本も輸入ものはなぜ高い

ある雑誌の8月号で面白いものを見つけた．国内で購入できるDOS/V対応マシンの紹介が一斉に載っていたのだが，不運（？）なことに，ある同じマシンの日本価格と米国価格が違うページに書かれていたのだ．「米国価格4,395ドル，日本価格922,000円」，1ドルを140円で換算すると，その差約30万円．さっきの話だと，もう1台i80486マシンが買えてしまう．サポートなどの問題もあるとは思うが，これはちょっと価格設定に問題があるんじゃないのかな!?

考えてみればこの業界，輸入物は非常に高い．まるで外国製品を拒むかのようにさえ感じられる．ソフトウェアなど，日本語版に移植している場合はいざ知らず，単純に輸入して販売するだけのものでも，1ドル＝200円程度の換算ではないだろうか!!　1ドル＝140円とまではいわないが，もう少し何とかならないものだろうか……．

そうそう，書いている間に思い出した．書籍の輸入物もなんとかして欲しい．今時，1ドル＝300円という換算では世も末である．



Computer Laws

コンピュータの法律

プログラムの改造・その3

熊谷信太郎（弁護士）

■改造禁止の特約

今回は，前回に引き続いてプログラムの改造に関する諸問題を取りあげたいと思います．

市販のプログラムのパッケージ上に，パッケージを破ったら購入者とプログラム著作権者との間で特約が成立したものとみなすという記載があり，その特約の内容として，プログラムの改造禁止があげられている場合には，パッケージを破った時点で，購入者と著作権者との間で改造禁止の特約が有効に成立するということは，前回説明しました．

ところが，そのような改造禁止という法的効果を十分に認識しないで，購入者が勝手にプログラムを改造してしまった場合にはどうなるのでしょうか．

著作権法の規定からすると，プログラムをより効果的に利用する目的で改造することは，もともと認められているのですから，著作権者の同意を得ないでプログラムの改造をしても，著作権法違反になることはありません*1．

■特約は契約の一部

しかし，プログラムの改造禁止の特約は有効ですから，特約違反，すなわち著作権者とプログラム購入者との間の契約違反の問題は出てきます．

ところで，契約という用語は，日常生活の中でも多用されますが，法律的な性質としては，個人や法人間の合意により成立する法律行為であり，契約当事者を合意内容により拘束するところに意味があります．また，特約というのは，一般的な契約内容に加えて特別な合意を契約当事者間で行うものであり，法律的には契約の一部を構成するものです．

このように，契約には拘束力があるので，いったん契約が成立した後で一方的にそれを取り止めることは，法定の契約解除事由があるなどの場合を除いては，原則としてできません．

また，一方の契約当事者がそれを守らなかった場合には，債務不履行と呼ばれ，その場合のペナルティとしてどのような効果を発生させるかは，例えば違約金やプログラムの使用の差し止めなどを，特約自体で定めていることが一般的です．その場合には，契約に定められた違約金を著作権者に支払わなければなりません．

もし，特約の中で債務不履行の効果を定めていなくても，民法が定める要件を満たせば，民法上の債務不履行の効果が発生します．それは，著作権者に発生した損害の賠償義務と契約の解除権発生です．そして，そのような効果が発生するための要件としては，承諾を得ないでプログラムを改造したため著作権に損害が発生し，それについて，プログラム購入者に故意，過失があり，かつ，その債務不履行が違法であることなどがあげられます．

プログラムの改造禁止の特約も有効な契約ですから，それに違反した場合には，通常はこれらの要件を満たし，債務不履行となり，著作権者から請求があれば，著作権者に生じた損害を賠償しなければなりません．

■改造が必要なときは

しかし，具体的な事案によっては，そうともいえないケースがありうるでしょう．たとえば，購入したプログラムでは購入者のコンピュータがうまく動かないという場合には，当然改造の必要性が出てくるわけですから，その場合には購入者としては，著作権者に同意してもらって改造するか，あるいは著作権者側に依頼してプログラムを改造してもらわないと，プログラムを購入した意味がありません．

しかし，そのいずれも拒否された場合には，購入者としてはそのプログラムでコンピュータを動かすためには，著作権者の承諾が得られなくても，自らプログラムを改造せざるをえない立場に追い込まれるのですから，そのような場合にまで，債務不履行責任を問うのは公平でないといわざるをえません．

したがって，そのような場合には，

ワンラインNEWS
ロゴジャパンは同社の「ロゴライター」を大幅に機能強化させた「ロゴライター2」を発売する．同ソフトは「CEC-Logo'90」に完全準拠している．

特約違反ではあるが，違法な債務不履行ではないとするか，あるいは，そのような不公平な内容の特約は公序良俗に違反し，その限りで無効になると考えて，プログラム改造者は損害賠償義務を負わず，解除権も発生しないと考えるべきでしょう*2.

＊1　著作権法第20条第２項第９号
＊2　民法第90条

アキバ短信

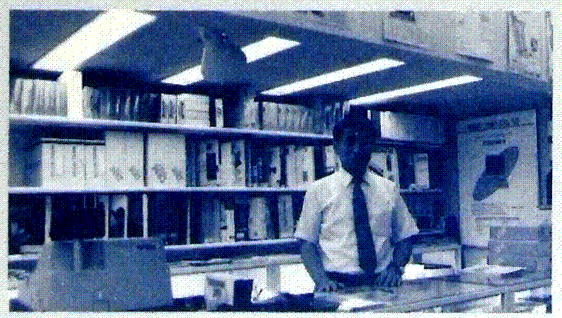
コムサテライト

PCが本場並みの低価格で手にはいる!

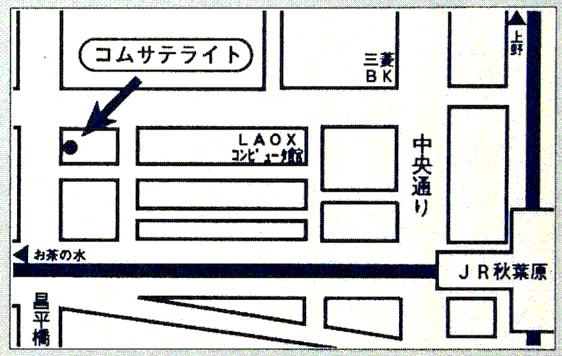
就職戦線はもはや終盤を迎えているらしい．大学生はみな，週末も出かけずにじっと企業からの呼び出し電話を待っているという．ただ待つだけではない．あの会社では何日ごろ内定が出るらしい，今週末には一斉に呼び出しがありそうだ，といった情報（というとカッコいいが，要は単なるウワサ）が友人どうしの間で電話で交換されているという．１社が内定を出せば，直ちに学生から学生へ，そして他社の採用担当者へと伝播する．まさに悪事は千里を走るのである．

パソコン通信もまた，ウワサを伝播するメディアとして，強大な威力を発揮しつつある．めずらしいものや，何か新しいものを見つけると，人間というのは誰かに話したくなるものなのだ．「新しいIBM PCのショップができた，しかもソフトがとっても安いらしい」というウワサが，日経MIXやNIFTY-Serveなどでささやかれている．それが今回紹介するコムサテライトだ．

コムサテライトは，IBM PCとその互換機のハードウェア，ソフトウェアを取り扱っており，店内には，PS/55Zや各種のボード，パッケージソフトなどが並んでいる．一見，ほかのショップとなんら変わりはないのだがコムサテライトの特徴は丁寧な技術サポートである．DOS/Vの登場で，



営業時間　10：00～19：00　木曜定休
TEL　03-3257-9431



漢字ROMを備えていないIBM PC互換機でも日本語表示が可能となったこと，IBM PC互換機は国内メーカーのパソコンよりもはるかに価格が安いことは，本誌の読者ならすでにご存知だろう．だが，いざ購入するとなると，故障の際にどの程度保証が受けられるのか，周辺機器との接続などで問題が生じたときにはどうしたらいいのか，といった不安で二の足を踏んでしまいがちだ．コムサテライトでは，こういう人のために「コンピュータクリニックサービス」を行っている．簡単な質問からハードウェアの持ち込みまで何でも受け付けてくれるので心強い．

また，注文に応じてキットを組み立てたり，独自のSateliteブランドの互換機も販売している．PC互換機というと，無骨でやたらと大きな匡体のものを想像しがちだが，Sateliteは厚さ５センチほどのスリムな形態で，しかも386SX 20MHzでも30万円未満だというから驚きだ．各種輸入ソフトウェアも通貨レートに変動はあるものの，1ドル約150円程度の換算率で販売している．安さの秘密は世界各国に張り巡らした情報ネットワークのたまものだそうだ．

コムサテライトは，秋葉原のメインストリート中央通りより１本お茶の水寄りの大通りに面した，神田明神下交差点の昌平橋よりにある．PCフリークならずとも一度はのぞいてみたい．

●プログラマの仕事場 | 6

Programmer's Working Space

アンテナハウス株式会社

製造から流通，販売まですべてを自社でまかなえるソフトウェアメーカーをめざす

アンテナハウスはWindows上でATOK6/7を利用するWinATOK/MSKNやMS-Works対応のファイルコンバータ，富士通，シャープ，リコー，キヤノン，松下，NEC，エプソンなど各社の専用ワープロとMS-DOS用ワープロソフト間のファイルコンバータのシリーズ化など，ソフトウェアを使う側の立場にたった便利でユニークな製品を次々と送り出している会社だ．同社をたずねて小林徳滋社長にお話をうかがった．

ソフトウェアの出版社としてパッケージソフトを企画・出版

「アンテナハウスという会社はソフトウェアハウスではありません．ソフトウェアメーカーです」と小林社長は語る.「基本的に製品を作って売っている会社です．ソフトウェアの出版社と考えていただければいいでしょう．マーケットにあう製品を作って，開発をして，それを販売しているのです．ですからソフトウェアの注文を受けて，ソフトを組んでお客さんに納品するという仕事はやっていません．うちの場合，一番たいせつなのは，製品を企画することなのです．どういう製品がこれから必要とされるだろうかということを考えて，そういう製品を作る方は社内でやることもあれば，外部で作ってもらうこともあります．外部のプログラマとか外部の開発会社に委託をして作ってもらうことのほうが多いですね」と小林社長．

同社の扱う製品は，自社企画自社開発のもの，自社企画で他社開発のもの，他社で開発した製品の販売権を同社が保持しているものの3通りがあるという．たとえば「WinATOK/MSKN」の開発元は本誌4月号のこの欄でも取りあげたソフトヴィジョンである．

独立系のソフトウェア会社というと，ソフトウェア技術者が自ら設立し，社内での開発が前提となっているところも多い．アンテナハウスのように「ソフトウェアの出版」を標榜しているところはめずらしい．それには，会社の設立の際の事情も反映されているようだ．小林社長自身，アンテナハウスの設立以前，ある新聞社系の出版社に籍を置いていたのだという．「(その新聞社の）グループ内でソフトウェア会社を作ろうという話がもちあがって，私はその企業化調査を担当しました．結局そのときはやめようという結論になったのですが，結果的にはそれを自分でやることになりました．現在の形はそのころから考えていたものです.」

各社の製品に対応した文書互換ソフトウェアを提供

アンテナハウスの製品群の中で，まず目を引くのは各社のワープロ専用機やワープロソフトに対応した文書互換ソフトウェアである．

ワンラインNEWS
アップルコンピュータはマルチメディアパソコンに対応した新しいシステムソフトウェア「クイックタイム」を発表した．出荷は今年末になる予定．

「文書互換ソフトウェアを最初にやったときには，MS-DOSのテキストファイルくらいしか交換する方法がなくて，しかもワープロ専用機はDOSのテキストファイルをほとんど変換できませんでした．変換するソフトを作っている会社もありましたが，その場合にはひとつのワープロ専用機とひとつのパソコンの間の接続しかやらないのです．アンテナハウスでは，それを品ぞろえして各社のワープロ全部に対応できるようにしようと考えて，積極的に進めてきたのです．でも，MS-DOSのテキストファイルでできることには限度がありますから，これらのシリーズは'89年で作るのはやめてしまいました．いま作っているのは，リッチテキストコンバータです．MS-DOSのテキストファイルでは文字しか移せませんから，いくらがんばっても限界があります．お客さんの要望はやはり罫線とか，書式なども全部もってきてほしいということですからね．」

最初は罫線や書式などを全部表現できるワープロ用の中間ファイルフォーマットを作り，この形式を介してワープロ間の変換を行うつもりだった．そこで，ファイルフォーマットの開発をソフトヴィジョンに委託したのだが，その後，アメリカのマイクロソフト社にリッチテキストフォーマットという標準フォーマットがあることを知って，マイクロソフト社と協力してこれを日本語化することとなったのだそうだ．

現在すでに，リッチテキストフォーマットを介した各種コンバータを発売中である(Products欄参照)．この製品群の特長は，ワープロまたはワープロ専用機とリッチテキストフォーマットの相互変換モジュールさえ作っておけば，組み合わせ次第でさまざまな製品となることだ．これは，変換エンジンと呼ばれる変換処理を行う部分，その中でも各ワープロに共通の部分，ユーザーインターフェース部分など，モジュール化を徹底して行うことで可能となった．このことは，ソフトウェアの制作進行という面でもメリットがあるという．最終的な製品の組み立ては社内で行うことにして，モジュール単位で社外に委託することができる．共通ライブラリや共通のコーディング上のマナーなどを決めておけば，社内でコントロールしながら外部のマンパワーが利用できるのである．

ユーザーの立場にたって本当に必要な形の製品を

アンテナハウスの商品企画の考え方について，小林社長は次のように語ってくれた．「コンバートソフトなどはそれぞれ1本単位で考えていきますとあまり商売にはならないのではないかと思います．ですから私どもでは，シリーズ化をかなり意識しています．ソフトウェアを作るときに1回で終わるものではなくて，できるだけ長期的にできるものを考えています．」

「今作っているこのEVERY-PCは丸2年以上かけています．以前はPA-PC98という名前でしたが，バージョンアップで名前も変えました．電子手帳とパソコンとの接続というのを，1番最初に企画したのはHAL研究所ですが，2番目はアンテナハウスでしょう．'88年の秋にシャープへ話をもち込んだところ，もう1社来ているといわれまして，それがHAL研究所さんだったのです．'89年にシャープはICカードのフォーマットを公開して，その後

About Company

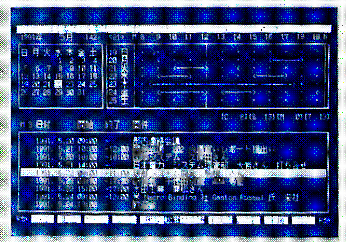
小林徳滋社長．1984年8月設立．コンピュータソフトの企画・開発と販売を行う．ソフトウェアの受託開発業務はいっさい行わず，自社でパッケージソフトウェアを企画し，社内外のプログラマを使って開発，販売したり，ほかのソフトウェア会社の開発したソフトウェアを販売する．ソフトウェアの「メーカー」を目指す．現在社員10名，うち5名がプログラマ．

Location

本社は千代田区麹町．新宿区市ヶ谷にはプログラマの勤務する開発室，長野県伊那市に伊那支店（商品センター）がある．

Products

「パソコン手帳 Every-PC Ver.2.0」(18,000円)
メモ，スケジュール，住所録などの個人情報のデータ管理ソフト．あらかじめ登録されたフォーマットを使用すれば，面倒なフォーム設計なしにその場で利用できる．別売のケーブル（15ピン8,000円，4ピン7,000円）を使ってシャープ製の全電子手帳と双方向のデータ交換が可能．他社ソフトに付属するケーブルも利用できる，電子手帳の絵文字の表示，印刷が可能などきめ細かい対応が特長．



Every-PC Ver.2

「リッチテキスト・コンバータ」(各種24,000円～29,000円)
日本語リッチテキストフォーマットを使ったワープロ同士の文書コンバータ．各ワープロの独自形式のファイルをリッチテキストフォーマットを介して変換するため，罫線や文字修飾などの情報も含めた変換が可

ワンラインNEWS
ダイワコンピューテックはアップル社のパソコン「Macintosh」に接続できる水平プリンタを発売した．プリンタは沖電気製で，独自のアダプタを装備した．

RUPO←→一太郎 Ver.3

能．MS-DOSテキストファイル，リッチテキストフォーマットの入出力もできる．現在「一太郎Ver.3←→OASYS30」「OASYS30←→新松(Ver.2.0)」「一太郎←→新松」「RUPO←→一太郎Ver.3」「RUPO←→新松」を発売中．「WinATOK/MSKN Ver.2」(9,800円)

MS-WindowsやMS-KANJI API対応の各種アプリケーションソフトでジャストシステムの日本語FEP，ATOK6/7を使えるようにするユーティリティ．WinATOKはMS-Windows2.11上で，MSKNは，MS-Windows3.0，MS-Works，Multiplan，QuickC，QuickBASICなどでATOK6/7を使用可能にする．

RTFJ(Rich Text Format Japanese)

日本語リッチテキストフォーマット．RTFJは日本語ワープロ用の文書ファイルの共通フォーマットである．RTFJのもととなったRTFは米国のマイクロソフト社が同社のワープロソフトMS-Wordとほかのソフトとの文書の互換性を保つために考案したもの．RTFJは日本語を取り扱えるようにRTFの仕様を拡張したフォーマットで，マイクロソフト，アンテナハウスの2社によって共同開発された．アンテナハウスでは，RTFJの仕様書，解説書，各種ワープロおよびワープロソフトとRTFJ間のファイル変換ツールをセットにした「RTFJ評価キット V2.0」(50,000円）も販売している．

に通信の企画を公開しました．'88年当時シャープから『いずれ規格を公開するので待ってほしい』といわれたので，私はおとなしく待っていたのですが，HAL研究所は独自に開発をすすめて'89年の7月には製品を出しました．うちは遅れて'90年の1月にVer.1，8月に1.1，そして今度Ver.2を出します．この製品に関しては，2年間もかけてきましたし，製品のでき具合には絶対に他社には負けない自信があります．うちではソフトウェアを長期的に改良しようと考えているのです．その点で，アシストさんなんかとは，全然考え方が違いますね．電子手帳の分野というのは，一時期かなり多くのソフトハウスが参入していました．大きいところはそうでもありませんが，小さなソフトハウスはパッと手を出して，すぐに手を引いてしまうことがあります．これはソフトハウスの悪い点ですね．やはりバージョンアップして長期的にサポートしていくというのがたいせつです．」

商品化は長期的に改良していけるものを提供

アンテナハウスの製品企画の担当者はソフトウェア技術者ではない．製品の仕様決定については，あくまでユーザーの立場に立って細かい点まで配慮している．たとえば同社の製品EVERY-PCでは，他社の電子手帳―PC接続ソフトのケーブルもすべて利用できる．もちろん，EVERY-PC専用のケーブルもあるが，これはソフトとは別売，つまり買っても買わなくてもよい．このような形にしたのもユーザーが以前に購入したものを無駄にせずにすむようにとの考えからだ．小林社長は，「こうしておかないとお客さんに迷惑がかかりますから」と語る．

また，このEVERY-PCはシャープ製の全電子手帳で利用可能で，外部インターフェースも4ピンと15ピンの両方に対応している．ユーザーが電子手帳を新製品に買い替えても，EVERY-PCはそのまま使えるというわけだ．同社では，電子手帳通信モジュールとEVERY-PCの本体プログラム間でデータ交換するためのインターフェースの標準規格を策定している．このため，通信モジュールさえ切り替えれば，本体プログラムは共通のままで各種電子手帳に対応できるのである．

ユーザーの立場になって新たな商品を企画してきた同社が，今後どのような製品を提供してくれるのか，非常に楽しみである．

Every-PCのデモ

ワンラインNEWS
マッジネットワークスはトークンリングに対応したパソコン用LANボードを拡販する．国内ではLANの規格はイーサネットが90%以上のシェアをもっている．

information・page

イベント

■9月4日(水)~9月6日(金)
データベース'91 TOKYO
商用データベース、インハウス・データベース用の各種システムの展示。
国際シンポジウムも開催される。
会場：池袋サンシャインシティ・センターホールA(東京)
入場料：500円
問い合わせ先：日本経済新聞社事業局　TEL 03-3252-8157

■9月5日(木)~9月8日(日)
1991年電子情報通信学会秋季大会
一般講演と通信・エレクトロニクスなどのシンポジウム講演が行われる。
会場：玉川大学(東京・町田市)
参加費：1,000円(会員)，1,700円(非会員)，500円(学生)
問い合わせ先：(社)電子情報通信学会　TEL 03-3433-6691

■9月25日(水)~9月27日(金)
第6回IC・光カード&ASCAT展
ICカード、光カードの最新技術と関連機器の展示。
セミナー、「SCAN-TECH JAPAN '91」も同時開催される。
会場：東京流通センター・第2展示場(東京・平和島)／入場料：1,000円
問い合わせ先：日本工業新聞社事業部　TEL 03-3292-3561

■10月1日(火)~10月5日(土)
1991年エレクトロニクスショー
国内外のエレクトロニクス関連製品、最新技術を展示。
10月1日~3日までは招待日、4日~5日は一般公開日。
会場：日本コンベンションセンター(幕張メッセ)
入場料：500円(一般)，300円(学生)
問い合わせ先：日本エレクトロニクスショー協会　TEL 03-3284-1051

■10月2日(水)~10月4日(金)
ソフトウェア・ショウ'91
ホビーソフトを除く，大型コンピュータからパソコンまでのソフトウェアを展示。
10月2日は招待日，3日~4日は一般公開日。
会場：池袋サンシャインシティ・センターホール(東京)／入場料：無料
問い合わせ先：ソフトウェア・ショウ事務局　TEL 03-3437-3394

■10月10日(木)
第40回オーディオフェア
コンピュータとシンセサイザーによるデジタルサウンド・コンテストが行われる。
作品の募集締切は10月3日。
会場：池袋サンシャインシティ(東京)
問い合わせ先：(学)尚美学園「デジタルサウンド・コンテスト」事務局 TEL 03-3818-7

■10月9日(水)~10月11日(金)
マルチメディア国際会議'91
マルチメディアに関する講演と関連製品の展示。
会場：日本コンベンションセンター(幕張メッセ)
受講料：85,000円(3日間共通，入場料・昼食代含)
問い合わせ先：マルチメディア'91運営事務局　TEL 03-3542-

海外イベント

■9月4日(水)~9月6日(金)
UNIX OPEN SOLUTIONS
UNIXの関連製品の展示とコンファレンスが行われる。
会場：San Jose Convention Center(米国・カリフォルニア)
問い合わせ先：ザ・インターフェイス・グループ・インク日本支社
TEL 03-3237-8759

セミナー

■10月8日(水)~10月9日(水)
オブジェクトオリエンテッドテクノロジー入門セミナー
オブジェクト指向による分析設計プログラミングからその長所短所までを解説する。
会場：麹町会館(東京)／受講料：96,000円(3名以上割引あり)
問い合わせ先：ソフトウェアテクノロジーインターナショナル
TEL 03-5689-3536

Microsoft University

■8月26日(月)~8月30日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース。
会場：特別会場T／受講料：180,000円(5日間)

■8月26日(月)~8月30日(金)，9月9日(月)~9月13日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
会場：三田会場／受講料：180,000円(5日間)

■9月2日(月)~9月6日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
会場：特別会場T／受講料：150,000円(5日間)

■9月9日(月)~9月13日(金)
MS OS/2 Presentation Manager Applications Programming
OS/2 PMアプリケーション開発のための上級コース。
会場：特別会場T／受講料：150,000円(5日間)

■9月17日(火)~9月20日(金)
MS OS/2 Programming Environment
OS/2アプリケーション開発のための基本コース。
会場：三田会場／受講料：120,000円(4日間)

■9月18日(水)~9月19日(木)
C言語入門
Quick Cを使ったC言語の入門コース。
会場：特別会場T／受講料：60,000円(2日間)

■9月24日(火)~9月27日(金)
MS OS/2 Device Drivers
MS OS/2デバイスドライバの機能と構造を解説し、サンプルの作成を行う。
会場：三田会場／受講料：150,000円(4日間)

■9月25日(水)~9月27日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
会場：特別会場T／受講料：90,000円(3日間)

Microsoft Universityは，OS/2，Windowsアプリケーション開発者のために定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウェア(株)MU教育センター　TEL 03-3452-77

NEWS & NEW PRODUCTS

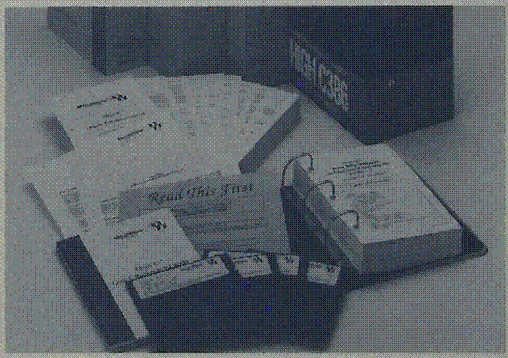
日本IBM，大容量メモリに対応した「DOS 5.0」を発表

日本アイ・ビー・エム（株）（TEL 03-3586-1111）は，メモリを効率的に活用するための機能を特に強化した「DOS 5.0」を発表した．従来のDOSではメモリ域が最大640KBという制限があったが，ハイ・メモリ・エリアと呼ばれるメモリ域にDOSの一部を移動させることにより，大容量メモリを必要とするアプリケーションを効率的に稼働することができる．また，DOS自体の容量であるカーネル・サイズも縮小している．複数のアプリケーションを切り替えながら実行するタスク・スイッチ機能により，ひとつのアプリケーションを一時中断し，別のアプリケーションに移ることができ，ヘルプ機能の強化，インストールやバージョンアップの作業が容易になった．価格は，28,100円，アップグレードは14,500円．

また同時に，DOS/V対応の「Lotus 1-2-3 R3.1J」，「ル・クローンランタイム/Vバージョン1.3」，「WORDSTAR 6.0」なども発売した．

テックソフト＆サービス，4GBのメモリが使えるHIGH C386をバージョンアップ

テックソフト＆サービス（株）（コンピュータ事業部：TEL 0474-23-1322）は，80386CPUのプロテクトモードに対応し，386｜TOOLBOXの386｜DOS-Extenderと組み合わせることにより，4GBのメモリを使用することができるC言語コンパイラのバージョンアップ版「HIGH C386 ver.1.7」を発売した．新たに，80387/Weitek3167のほか，80486内蔵のFPUなどのコプロセッサをサポート．MASM，OS/2 Ver.2.0のOMFと同じフォーマットのオブジェクトを生成することができる．価格は，178,000円．



マイクロソフトウェア・システムズ，CD-Iタイトル開発ツールを発売

マイクロソフトウェア・システムズ（株）（TEL 03-3257-9000）は，CD-I，Cライブラリの機能を拡張し，CD-RTOS上で高水準なユーザーインターフェースアプリケーションを容易に開発できるマルチメディア開発ツール「CD-RTOS/RAVE」を発売した．ディスプレイ・スクリーン・マネージャとプログラム・サポート・ライブラリで構成されており，価格は，200,000円（OS-9用），230,000円（UNIX用／SUN-3，4・NEWS）．

東芝，大容量HDD内蔵の32ビットデスクトップパソコンを発売

（株）東芝（パソコン営業推進部：TEL 03-3457-2930）は，CPUに80386（25MHz）を搭載し，170MBのハードディスクを内蔵したデスクトップパソコン「J-3100ZD171モデル」と，100MBのハードディスクを内蔵した「同101モデル」を発売した．標準で4MBのメモリを装備し，グラフィック画面と文字画面を重ね合わせる「スーパーインポーズ機能」に加え，640×480ドットのVGAモードを標準でサポートしている．オプションの21インチ高解像度ディスプレイ（798,000円）と高解像度インタフェースカード（348,000円）を使用すれば，最大1280×1024ドットの表示が可能となる．また，日本語MS-DOS V3.1を標準装備している．

■おもな仕様

プロセッサ：80386（25MHz）／メモリ：4MB（最大14MBまで拡張可能）／外部記憶装置：3.5イン

ワンラインNEWS
日本IBMはパソコン上でCIMソフトを開発するための支援ソフト「プラントワークス」を発表した．このソフトの利用によりパソコンだけでシステムが構築できる．

NEWS & NEW PRODUCTS

チFDD×1・100MB HDD×1（101モデル，最大200MB），3.5インチFDD×1・170MB HDD×1（171モデル，最大340MB）／拡張スロット：J-3100カード専用×2，IBM PC/ATフルサイズカード用×3 ／価格：898,000円（101モデル），1,080,000円（171モデル）

日本電気，カラー液晶搭載の32ビット低価格ラップトップマシンを発売

日本電気（株），日本電気ホームエレクトロニクス（株）（NECパソコンインフォメーションセンター：TEL 03-3452-8000）は，4,096色中16色の同時表示が可能なTFT型カラー液晶ディスプレイを搭載し，85万円からという低価格を実現したカラーラップトップパソコン2モデル「PC-9801Tモデル F51」「同F71」を発売した．また，同時に100MBハードディスクを内蔵し，白黒液晶ディスプレイを搭載した「PC-9801TモデルW7」（750,000円），9.8インチTFT型外付けタイプのカラー液晶ディスプレイ「N5912-F1」（465,000円）なども発売した．

■おもな仕様

プロセッサ：80386SX（20MHz）／メモリ：1.6MB（最大14.6MBまで拡張可能）／外部記憶装置：3.5インチFDD×2・40MB HDD×1（モデルF51），3.5インチFDD×2・100MB HDD×1（モデルF71）／ディスプレイ：9.8インチTFT型カラー液晶／価格：850,000円（モデルF51），950,000円（モデルF71）

三菱電機，3モードFDDを搭載した32ビットノートブック型パソコンを発売

三菱電機（株）（情報処理機器事業部計画部：TEL 03-3218-3285）は，同社のAXパソコン「MAXYシリーズ」に，80386SX（16MHz）を搭載したノートブック型パソコン「MAXY NOTE 386」の新モデル「M3122-Bシリーズ」を発売した．この新モデルは，3.5インチの3モードフロッピーディスクドライブ（720KB/1.2MB/1.44MB）をサポートしており，AX機以外でフォーマットされたフロッピーディスクの読み書きが行える．既存の同M3122-Aシリーズと同様，着脱可能なハードディスクパック方式を採用．MS-DOS，MS-Windows，MAXYメニューなどがあらかじめハードディスクにインストールされている．ディスプレイ別売のタイプ（M3122-B12M/B14M）もある．

■M3122-B12/B14/B12M/B14M

プロセッサ：80386SX（16MHz）／メモリ：1MB（最大5MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・20MB HDD×1（B12/B12M），3.5インチFDD×1・・40MB HDD×1（B14/B14M）／ディスプレイ：バックライト付き8階調TST液晶（B12/B14，B12M/B14Mはディスプレイ別売）／価格：398,000円（B12），438,000円（B14），338,000円（B12M），378,000円（B14M）

アイ・オー・データ機器，98対応シリコンディスクボードほかを発売

（株）アイ・オー・データ機器（TEL 0762-60-3366）は，PC-9800シリーズ対応の超高速記憶装置シリコンディスクボード「SDBシリーズ」を発売した．1枚のシリコンディスクボード「SDB」に40MB（8MB容量のメモリモジュールを5枚搭載）が増設でき，予算に応じてメモリモジュールの増設が可能，最大512MBの大容量を実現することができる．ボード上にバックアップ電池（パソコン

NEWS & NEW PRODUCTS

使用中に充電）を搭載しているため，外部にバッテリをもつタイプにくらべて安全性が高い．価格は，110,000円（SDB-8M/8MB），190,000円（同16M/16MB），260,000円（同24M/24MB），340,000円（同32M/32MB），420,000円（同40M/40MB），85,000円（SDEX-8M/8MBメモリモジュール）．

また，同社ではエプソンPC-386シリーズ用内蔵メモリボード「EP-RB7シリーズ」を発売した．80386搭載機種用でベースボード上にSIMモジュールを２枚実装している「EP-RB7D-4M」(4MB, 62,000円)，「同8M」(8MB，100,000円）と80386SX搭載機種用でSIMモジュールを１枚実装している「EP-RB7S-4M」(4MB, 57,000円)がある．SIMモジュールの価格は，29,000円(EP-SIM-2M/2MB)，49,000円(同4M/4MB)．

アイシーエム，98NOTE対応のハードディスクパックを発売

(株)アイシーエム(営業部：TEL 06-648-4702)は，PC-9801NS/Eのハードディスクスロットに挿入するディスクパックタイプの40MBハードディスク「PACK E-40」を発売した．256KBのキャッシュメモリを搭載し，本体の環境設定でモーター制御やレジューム機能を設定して電力消費を節約することができる．出荷時に全領域がMS-DOSフォーマット済みのため，MS-DOSのシステム(Ver.3.3X)を転送するだけですぐに使用することができる．メニューソフト「EOシステム」などのユーティリティもインストールされている．価格は，118,000円．

エプソン，マッキントッシュ対応プリンタドライバソフトを発売

セイコーエプソン(株)，エプソン販売(株)（エプソンインフォメーションセンター：TEL 03-3377-3500）は，同社のプリンタ「エプソンーVP/HG/XP/FBPシリーズ」26機種でMacintoshの印字出力を可能にするプリンタドライバソフト「EPSON PRINT」と，プリンタドライバソフト，プリンタ用I/Fボード・ケーブルをセットにした「MAC用I/FセットA/B/C」を発売した．「EPSON PRINT」には，24ドット用プリンタドライバ２種と48ドット用プリンタドライバ１種の計３種のドライバが入っている．価格は，15,000円(EPSON PRINT)，20,000円(MAC用I/FセットA/B)，25,000円(同セットC)．

インターコム，通信ソフト「まいとーく」をバージョンアップ

（株）インターコム（TEL 03-3293-3335）は，パソコン通信をすべて自動化できるオートパイロット機能，同時に２つの処理を行うことができるマルチタスク機能などをサポートしたPC-9800シリーズ用パソコン通信ソフト「まいとーくVer.2」を発売した．従来からの基本機能も大幅にアップし，検索，置換やブロック編集など本格的なワープロ機能を備え，XMODEM，YMODEM，B Plusなど８種類のプロトコル，最大19,200bpsの通信速度をサポートしている．ディスクユーティリティもファイルのツリー表示やディレクトリの削除や変更が簡単に行えるなど，さまざまな機能強化が行われた．販売は(株)インターソフト(TEL 03-3293-3338)が行う．価格は28,000円．バージョンアップ料は，「まいとーくVer.1.2」「同mono Ver.1.2」ユーザーに対しては5,000円，「同Ver.1.0」「同mono Ver.1.1」ユーザーに対しては7,000円．

ワンラインNEWS
日本ビジネスシステムズは勧角証券，三洋電機と共同でICカードを差し替えるだけでホームトレーディングやパソコン通信のできる多機能電話端末を開発した．

NEWS & NEW PRODUCTS

インターコム，ファイル転送ユーティリティ「LAPLINK」の日本語化を完了

（株）インターコム（TEL 03-3293-3335）は，米国トラベリング・ソフトウェア社のベストセラー超高速ファイル転送ユーティリティ「LAPLINK」の日本語化を行い，PC-9800シリーズおよびJ-3100シリーズに対応した「LAPLINK-J」を発売した．「LAPLINK」は，2台のパソコン間で高速にファイル交換を可能にするソフトウェアで，付属の専用4ヘッドRS-232Cケーブルで結び，最高速度115,200bps，CRC-16方式により100%エラーフリーでファイルを転送することができる．また，「LAPLINK-J」は，リモートインストール機能をもち，ディスクドライブをもたないパソコンやメディアサイズの異なるパソコンに対して，接続先のパソコンからLAPLINK-Jプログラムを送り込んで起動することができる．IBMのSAAに準拠したユーザーインターフェースにより，マウスを使ったプルダウンメニュー方式でほとんどの操作が行える．販売は（株）インターソフト（TEL 03-3293-3338）が行う．価格は，28,000円．

アップル社・IBM社，1990年代に向けた技術構想を発表

7月3日，カリフォルニア州サンノゼで，米国アップルコンピュータ社とIBM社は，1990年代に向け，強力でオープンな新しいソフトウェア基盤の開発を協力して行うという覚書に調印したことを発表した．この構想は，次の4項目からなる．
①オープンな新しいソフトウェア基盤をオブジェクト指向技術に基づき開発し，使いやすさとシステムの機能性を高め，インテル社のX86，モトローラ社の680X0，IBM社のRISC System/6000 POWERアーキテクチャなど，業界の主要なハードウェアで稼働し，現行のOS用のアプリケーションも新しい環境下で稼動できるようにする．この構想を実現するため，共同出資による独立したソフトウェア会社を設立する．
②クライアント／サーバ型のIBMエンタープライズシステム環境下にMacintoshを統合するため，ネットワークおよび通信関連製品を開発する．IBMのオープンシステムであるAIX（IBMのUNIX対応OS）の機能強化を行い，MacintoshとOSF／Motif双方のユーザーインターフェースを提供する．
③IBMのPOWERアーキテクチャに基づく1チップ化されたチップを将来のMacintoshに採用する予定で，両社はワークステーションとファイルサーバにPOWER PCマイクロプロセッサを搭載する予定である．これに基づく新しいチップの設計は，米国モトローラ社とIBMが共同で製造を行う．
④マルチメディアの共通基盤となる，機種に依存しないソフトウェア環境を提供していく．

セガ・エンタープライゼスと日本ビクター，業務提携で基本合意

（株）セガ・エンタープライゼスと日本ビクター（株）は，両社のホームエンターテイメント，業務用アミューズメント分野，AVエレクトロニクス分野の各基幹事業を相互に融合し，新事業の展開をはかるために業務提携することで，基本的合意に達したことを発表した．ホームエンターテイメント，業務用アミューズメント分野を得意とし，ハード・ソフト開発力，事業基盤をもつセガ・エンタープライゼスとAVエレクトロニクス分野における技術開発力と事業基盤をもつ日本ビクターの両社の事業基盤を相互に融合させることによって，両社の得意とする事業領域に新たな付加価値を加え，将来のより活発な事業展開をめざす．業務提携にあたっては，特定分野に限定することなく，研究開発を進める．第1弾として，セガ・エンタープライゼス社の「MEGAドライブ」に，日本ビクターのCD-ROMドライブを結合した新商品を共同開発する予定．また，日本市場だけでなく，米国，欧州など海外においても業務提携を展開していく予定で，家庭用だけでなく，業務用の分野においても共同プロジェクトで事業を行うことを検討していく．

ワンラインNEWS
ジャストシステムは東芝のラップトップ型パソコン「J-3100」用にLAN対応の「一太郎 Ver.4 LANPACK」を発売．すでにPC-9800用は発売されている．

Programmer's Program

16機種に対応したスクリーンエディタをひとつのパッケージに

Final Super Pack（ファイナルスーパーパック）

ほとんどのMS-DOSマシンに対応

「Final Super Pack」は，ひとつのパッケージで日本電気PC-98/9800，エプソンPC-286/386，富士通FMR，同FM-TOWNS，同FM-16β，松下電器Panacom M，日立B16/32，同2020，東芝J-3100/3300，日本IBM PS/55，IBM PCおよび互換機，各社AXと，ほとんどのMS-DOSマシン，さらにシャープX68000に対応している．パッケージには5インチと3.5インチのフロッピーディスクが各4枚，バージョン3とバージョン4用のユーザーズマニュアルが各1冊，登録ハガキ，ディスク構成とインストール方法や機種ごとの特殊キーの違いを記した小冊子が入っている．

8枚のフロッピーディスクには，各機種用のFinal本体と環境設定のためのカスタマイズファイル，そして各種ユーティリティツールが収められている．ユーザーは，自分の使用機種や環境に合わせてインストールする必要がある．

なおインストールは，各機種に対応したFinalの入ったフロッピーディスクをドライブに挿入し，MS-DOSのコピーコマンドを使って必要なファイルをコピーするだけである．

Finalを起動するには，コマンドラインから

```
A> FE
```

と入力すればよい．編集ファイル名の指定なしでFinalを起動すると，まずファイラー（ファイル選択画面）が表示される．カーソルキーを使って目的のファイルに反転カーソルを移動し，リターンキーで選択する．指定されたファイルを読み込むと編集画面になる．ファイラーは起動時だけでなく，編集中に新たに別ファイルを読み込む場合にも表示される．編集キー操作はワードスターに準拠しており，またカスタマイズも可能なので，ほかのエディタから乗り換えた場合でもほとんど違和感なく使える．

多機種で共通した操作を実現

Finalの特徴的な機能は次のようなものである．

バイナリ編集機能

バイナリエディットモードと呼ばれる通常のテキスト編集とは異なった編集方法がある．バイナリエディットモードの編集画面では，16進数のコードが表示される．テキストの編集画面と同様にカーソルを操作し，16進数による入力で編集を行う．このモードを使えばデータファイルや実行ファイルなどを手軽に編集することができる（写真2）．

拡張子識別機能

FE.OPTで指定したファイル名の拡張子を識別して，ファイラーでのファイル名の表示色や右マージンを設定したり，自動的にバイナリエディットモードで立ちあげたりすることができる．

正規表現を使った文字列の検索と置換

Finalには正規表現を用いた検索・置換機能がある．以下の文字を特殊文字として，一般の文字列と組み合わせて検索する文字列を指定する．

. ^ $ # / [ ] * + ¥

この機能を用いることにより，複雑な文字列の検索や置換を行うことが可能になる．ただし，Finalの正規表現は全角文字には対応していないので，全角文字コードの2バイト目が正規表現の特殊文字と一致する場合などには，正常な動作を期待することはできない．全角の混ざった文字列を検索するには，起動時に正規表現を無効にする「-R」オプションを付加する．

ワード定義の変更

たいていのエディタは，編集中に[CTRL]+[A]や[CTRL]+[F]を使ってワード単位のカーソル移動を行うことができるが，Finalはこのワードの区切りを自由に変更することができる．デフォルトでは，スペースとタブがワードの区切りとして定義されており，後述のカスタマイズ機能によって，さらに「,」や「.」を区切りとして追加することができる．ただし，こ

ワンラインNEWS
知識計画はパソコン通信のホスト局を運用するためのソフト「BBS-Workshop Jr」を発売した．1回線対応の普及型でNAPLPS対応の画像データを扱える．

の機能も全角文字があると正しく動作しないことがある.

ヒストリー機能

起動時に-Hオプションを指定すればヒストリー機能が有効になる. 編集画面で入力した一連のテキストをFE.HST(オプションで任意のファイル名が指定可能) に圧縮した形で自動的に記録する. ヒストリーの読みだしは, 起動時-Rオプションを指定する. Finalは通常通り起動し, 編集画面が表示されたところで, FE.HSTの読み込みを開始し, キーボードから入力が行われたのと同様にテキストが編集される. 読み込みが終了すると, 通常通り編集操作ができるようになる.

この機能によって, 停電などで失われた編集ファイルを取り戻すことも可能である.

カスタマイズ

カスタマイズ機能により, コントロールコマンド, ファンクションキー, パーマネントマクロ, 短縮文字列, ワードの定義, 起動オプションを組み込むことができる.

まず, FE.CNFというカスタマイズの内容を定義するテキストファイルをFinalを使ってオープンし, 目的の機能を変更する. 次に, CONFE.EXEというカスタマイズツールを使ってFinal自身を書き換える. MS-DOSのコマンドラインから

A> CONFE

と入力すると, メッセージが表示され, コマンドラインに戻る. これでカスタマイズは完了である.

短縮文字列入力

[A] を入力してから [XFER] キーを押すと, カーソル位置に「argv」が挿入される. これは短縮文字列として [A] に「argv」があらかじめ登録されているためである. 短縮文字列は, すべてのキーにそれぞれ8個の文字列が登録されている. これらを任意の文字列に変更することも可能である.

ただし, この機能は日本語フロントエンドプロセッサの起動キーの設定によっては使えない場合がある.

ほかにもフリーカーソル, カット&ペースト, タグジャンプ, マーク行へのジャンプなどエディタとして必要な機能はひととおり揃っている.

現在, 市場にはDOS/Vを搭載したPS/55Zやテラドライブ, マルチメディアをターゲットとしたX68000やFM-TOWNSと多種多様なパソコンが存在している. 今後は, これら多くの異なった機種上でプログラム開発をする機会も増えることだろう.

プログラム開発の大半の時間を費やすエディタ, このエディタの操作性がプログラム開発の効率を左右するといっても過言ではない. その点, Finalはひとつのパッケージで16機種にも及ぶパソコンに対応しているので, どの機種でも同じ操作性を得ることができる. なお, UNIX上でも同様の操作性の得られるX-Finalも同社から販売されている.

■おもな特徴

コマンド入力方法：コントロールコマンド入力方式とポップアップメニュー方式／日本語処理：各種日本語フロントエンドプロセッサとの組み合わせが可能／編集サイズ：最大2MB／1行の長さ：ソフトウェアキャリッジリターンの挿入により最大2MB／同時編集ファイル数：最大10個／同時に開くことのできるウィンドウ数：最大2個 (PC-9801, PC-286/386は3個) ／コマンド体系：WordStarに準拠 (カスタマイズ可能)

■Final Super Pack

販売：（株）エー・エス・ピー（TEL.03-3767-1451）／価格：28,000円／動作環境：日本電気PC-98/9800・エプソンPC-286/386・富士通FMR・同FM-TOWNS・同FM-16β・松下電器Panacom M・日立B16/32・同2020・東芝J-3100/3300・日本IBM PS/55・IBM PCおよび互換機・各社AX・シャープX68000各シリーズ／必要メモリ：384KB以上／動作OS：MS-DOS Ver.2.XX以上(FM-16βはVer.3以上, X68000はHuman68Kに対応)

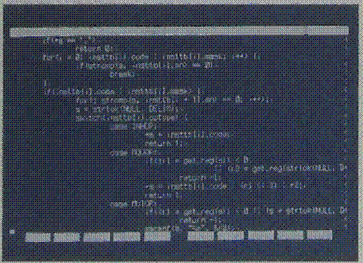


写真1　テキストエディットモード

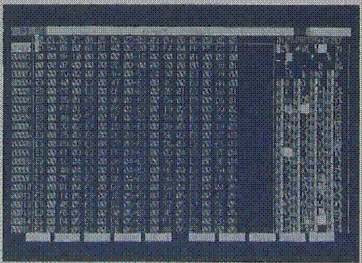


写真2　バイナリエディットモード

オンラインソフト採集帳 9

データファイル実行ツール EXEXE.EXE

通常，MS-DOS上では実行形式ファイル(COM，EXE，BATの拡張子をもつファイル）以外のファイル名をコマンドラインから入力しても，「コマンドまたはファイル名が違います.」というメッセージが虚しく返ってくるだけで，実行することはできない.

「EXEXE（「えぐぜぐぜ」と読む)」は，コマンドラインから入力されたデータファイルなど，実行できないファイルを実行する常駐プログラムである．ただし，実際にデータファイルを実行することはできないので，あらかじめ各拡張子に対応させた実行形式ファイルを呼び出すことになる.

まずEXEXE.CFGの設定を行う

EXEXEを使うには，LHAを使って適当なディレクトリに復元し，エディタを使ってEXEXE.CFGの内容を書き換える必要がある．EXEXE.CFGの内容には，1行に拡張子とその拡張子をもつデータファイルに対応する実行形式ファイル（拡張子がCOMまたはEXE)，そして実行形式ファイルに渡すパラメータを区切って記述する．また「#」ではじまる行は，コメントとして無視される(リスト1)．実行形式ファイルには，入力したファイル名のほかに環境変数やパラメータを渡すことができる（リスト2).

デフォルトでは，実行形式ファイルの検索パスが環境変数PATHに固定されているが，コマンドラインより入力されたコマンド名にパスの指定がなく，かつ拡張子が明示されていた場合にかぎり，その拡張子に固有の検索パスでコマンドを検索させることができる（-eオプション指定時のみ)．なお，複数の検索パスを指定する場合は，「;」で区切るか，複数の行に分けて記述する.

EXEXEを起動する

EXEXE.CFGの設定が終わったら，MS-DOSのコマンドラインから，

```
A> EXEXE
```

と入力すれば，メッセージを表示した後，再びコマンドラインに戻る.

リスト1

```
#       EXEXE sample configuration file

#       アーカイバ

.lzh    lha.exe         x -m $* $0
.ish    ish.com         $< $0
.zip    pkunzip.exe     $< $0
.arc    pkunpak.exe     $< $0
.zoo    zoo.exe         -e $< $0
.arj    arj.exe         -e $< $0
.pak    pak.exe         -e $< $0

#       画像ローダ

.q4     xld4.exe        $0 $*
.gra    qld.exe         $0 $*
.mki    magd.exe        $0 $*
.mag    magd.exe        $0 $*
.pic    picwl.exe       $< $0
.gif    gif.exe         $< $0
.rl     gl.com          $*
.gl     gl.com          $*
.bl     gl.com          $*

#       そのほか

.bdf    bupdate.exe     $0 $<
.dev    adddrv.exe      $<

.*      mifes.exe(¥)    $0 $>
```

EXEXE.CFGの設定がまちがっていると，ここでエラーが表示さる．また，実行形式ファイルが見つからないようなときには，ワーニングが表示される．このとき，実行ファイルが環境変数PATHで指定されていないディレクトリにある可能性があるので，パスのチェックが必要である.

起動オプションには，-r，-n，-?などがある．たとえば-rを指定すれば常駐解除，-nでEMSメモリの使用を禁止，-?でオプションパラメータの一覧が表示される.

たとえば，EXEXE.CFGの中でGIFの拡張子に対して実行ファイルGIF.EXEを指定して，一度EXEXEを実行すれば，MS-DOSのコマンドラインから,

```
A> TEST.GIF -A
```

と入力するだけで,

```
A> GIF TEST.GIF -A
```

と入力した場合と同じ処理が行われる.

FDと組み合わせればさらに便利

EXEXEは，ファイル管理ユーティリティと組み合わせることによって，より効果的に使うことができる．たとえば，FDの場合は目的のファイルにカーソルを合わせ，[X]と入力しリターンキーを押すだけで，EXEXE.CFGで指定したファイルが実行でき

リスト2

```
$<       入力ファイル（フルファイル名）
$*       入力ファイル（フルファイル名から拡張子を除いたもの）
$:       入力ファイル（パス）
$.       入力ファイル（フルファイル名からパスを除いたもの）
$&       入力ファイル（フルファイル名からパスと拡張子を除いたもの）
$>       入力ファイル（フルファイル名，ディレクトリだった場合は末尾に「¥*.*」が付加される）
$0       入力ファイルに指定された全パラメータ
$n       入力ファイルに指定されたn番目のパラメータ
$$       「$」自身
$(env)   環境変数env
${env}   環境変数env
$env     環境変数env（環境変数名として英字のみ使用可能）
```


ワンラインNEWS
富士通はVANサービス「FENICS」でファクシミリ配信サービス分野に参入した．特徴は漢字コードの違うコンピュータからのデータの受信を可能にしたこと.

ることになる．ほかのファイル管理ユーティリティでも，外部プログラムの実行をサポートしていれば同様に使用することが可能である．

画像ファイルを表示しようとすると，従来では各形式ごとに複数のグラフィックローダーを選択し，起動しなければならなかった．しかし，EXEXEを使うことによって，ファイル名をコマンドラインから入力するだけで，画像データを見ることができるようになる．Macintosh OSやMS-Windowsなどでは，データファイルをマウスでダブルクリックすれば，当然のことのように関連する実行形式ファイルが立ちあがる．グラフィックを多用した視覚的なシェルとコマンドラインという大きな違いはあるものの，このあたりまえの機能がMS-DOSでもやっと実現されたという感じである．

■EXEXE Ver. 2. 30

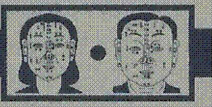
名称：えぐぜぐぜ／分類：常駐ユーティリティ／登録名：EXEXE230.LZH／作者：Ksoq（ASCII - NET PCS pcs31392）／動作環境：MS - DOS Ver.3.xxが動作するMS-DOSマシン／登録月日：1991年7月1日／復元方法：LHA E EXEXE230／入手場所：ASCII - NET PCS（pool msdos #4106）

ファイル構成

Listing of archive : EXEXE230.LZH

Name	Original	Packed	Ratio	Date	Time	Attr	Type	CRC
EXEXE.CFG	349	206	59.0%	91-02-02	22:59:40	a--w	-lh5-	4442
EXEXE.DOC	8299	3832	46.2%	91-06-29	21:16:18	a--w	-lh5-	EBF1
EXEXE.EXE	40666	21928	53.9%	91-06-30	21:22:08	a--w	-lh5-	F87F
EXEXE2.CFG	1644	566	34.4%	91-07-01	18:06:40	a--w	-lh5-	593A
UPDATE.DOC	3555	1484	41.7%	91-07-01	21:29:22	a--w	-lh5-	C139
EXEXE.GID	12626	5235	41.5%	91-06-30	21:53:38	a--w	-lh5-	9F7C
EXEXE230.SRC	42044	42044	100.0%	91-06-30	21:24:18	a--w	-lh0-	2338
EXEC#!17.LZH	11805	11805	100.0%	91-07-01	18:13:38	a--w	-lh0-	EB69
8 files	120988	87100	72.0%	91-07-01	23:15:46			

★UPDATE INFORMATION★「FD」Ver.2.32に（7月5日），「WTERM」がVer.8.50に（7月9日）「LHA」がVer.2.13に（7月20日）

HEALTHY?  UNHEALTHY?

⑨ なぜ砂糖は歯に悪い？

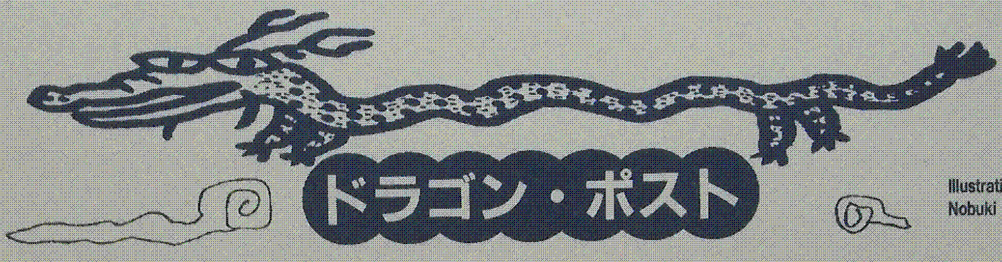
「甘いものばかり食べると虫歯になる」とは，子供のころよくいわれたこと．これは，テレビCMでおなじみのミュンタース菌などの口内菌が，糖分を分解する過程で酸を発生させ，その酸が歯を侵食するのが理由である．ここで問題なのは，糖分の種類だ．口内菌はとくに蔗糖，つまりグラニュー糖や白砂糖などとして売られている普通の砂糖が好物なのだ．清涼飲料水などによく用いられるブドウ糖や果物に含まれる果糖などは，虫歯の原因となる酸をあまり発生させない．もともとは飴の原料である麦芽糖も，比較的害がないのだが，市販のキャンディ類は砂糖を原料にしたものが多い．

次に問題となるのは，砂糖が含まれた食品の「粘りけ」だ．粘りがあり砂糖を多く含んだ食べものほど，歯にまとわりついて，口内菌の温床となってしまう．たとえば，チョコレートのような食べものがそれにあたる．「砂糖の入った食品を食べたら，よく歯を磨いてうがいをして」という，教科書的な対処法しかない．

歯並びも虫歯など歯の病気に関係している．歯並びがガタガタしていれば，食物がつきやすく，虫歯になりやすくなる．この歯並びは，顎の骨の大きさと関係している．顎の骨が小さければ，歯が納まりきらず，いわゆる八重歯や乱杭歯になってしまう．

歴史的には，人類の顎の骨はどんどん小さくなっているのだという．次第に柔らかい食物を食べるようになったからだ．恐いのは美的感覚もそれにともなって，顎の小さいほうが美しいと思うようになったこと．松田聖子のように「八重歯が可愛い」タレントは数多くいる．下顎が発達していたりすると「ゴリラのよう」といわれることさえある．顎に納まりきらなくなった歯がどうなるかといえば，顎に合わせて歯の本数が減ってしまうらしい．写真家の藤原新也氏が松本伊代の歯の数が少ないと指摘したのは80年代半ばのことだったが，現在では歯の少ない子どもが増えているのだろうか．

ワンラインNEWS
クボタコンピュータはグラフィックスコンピュータ「TITAN」ファミリーの卓上型ワークステーション「Vistra」シリーズを発売した．40 MHzのi 860を採用．



ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成しています．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★月日がたつのは早いものーその1

30代半ばのおじさんです．コンピュータとのつき合いは永く，17～18年前，NECの汎用機を仕事の合間に使わせてもらったのが，初めてでした．当時はFORTRANでカードに穴を空けてプログラムを作りました．今思うとなつかしいというか，世の中の時計の進み方が早いというか，隔世の感がありますね．このまま進めば声でプログラミングできる時代も遠くないかな？（小谷地和博）

――これはなつかしい．私がはじめてコンピュータに接したときは紙テープを使っていましたね．学祭では，この紙テープでしおりを作って配ったものです．案外人気があるんですよね，これが．

★僕が就職するまでには……

学生という立場上，「職場」の形態に興味を持っている．とくに「ソフトウェア」関連の人達の中には，相当追われているんだなと思わせる文章を書く人がいます．大変だなあ．それにしても，プログラムリストは「作るもので，読むものではない」と思わせるほど読みにくいですね．ハードウェア直接操作もの(デバイスドライバなど)は仕方ないとしても，いわゆる「アルゴリズム」という？プログラムの大まかな動作部分は，理想的にはグラフィカルな方が見やすいと思います．でも，そんな環境が実現するのは何年先になるのやら……僕が就職するまでには無理でしょうか？（大原和孝）

――プログラムの流れをグラフィカルに示すものとしては，フローチャートやPAD図などがありますね．今でも，PAD図で記述したプログラムを直接コンパイルするようなものはあるようですよ．

★男はだまってアセンブラ／

巷ではC言語が流行しているが，C言語よりもアセンブラだ／　C言語で用意されていない関数があったらどうする？　やはりアセンブラで書くしかないのだ！　アセンブラは速いし，細かい芸もできるし……．男はだまってアセンブラをやるべし／　もちろん女だってアセンブラ／　C言語をやっていいのは，ニューハーフだけだ／というわけで編集部の人おねがいします．（石橋亨）

――なるほど，そうですか．やはり世の中には“アセンブラ命！”という人がいるんですね．たしかにアセンブラで書かれたプログラムは小さいし早いけど，Cは高級アセンブラだという人もいるし……．

★付録フロッピーがいけない

ハードディスクを入れたとたん，ガチガチのアプリケーションユーザーになりさがってしまいました．プログラムを少しずつでも積み重ねていきたいと思っていたのに，なんたるザマなのでしょう．そうです．貴誌の付録フロッピーがいけないのです．LHaで圧縮・復元をし，毎日FDでファイルの整理に血道をあげている，そんな自分に嫌気がさしてしまいます．ここらで何か1つ作らねば，と思う毎日．（石川隆一）

――そういうときは，迷わずMS-C 6.0をフルインストールすることをお勧めします．こうすれば，ハードディスクの中は開発環境オンリーになりますから，安心してプログラミングに励めますよ．

★宝の山，発見／　これぞハード屋の醍醐味

今月のアキバ短信は実におもしろい記事でした．無造作に置いてある部品や機器が単なるガラクタに見えるか，宝の山に見えるかは本人の技術力しだいですね．おっ／これは○○に使えるというものを見つけたときは，実に気分がいいものです．これぞハードの醍醐味．こんな時つくづくハード屋でよかったと思ってしまいます．（JR8FVN/1）

――まともなものさえガラクタにしてしまいかねない素人にとって，ジャンクを使いこなせる人はほんとうに羨ましい存在です．あれはもしかしたら使えるんじゃないかなぁ～と思いつつ見送る毎日です．

★大きなプログラムを作ってしまったら

異機種パソコン間の伝達によるデータ交換ツールがぼちぼち増えてきた．メガソフトのMAXLINKが老舗であるが，NBLINK，転送くんなど，より多機種に対応できるものが増えている．HDD上で数MBを消費してしまうようなプログラムを作ってしまった誰かさん（誰でも？）には，便利なものではなかろうか？（ちなみに私は転送く

んのファンです。)（長谷川直樹）
——MAXLINKは編集部でも利用していますが，ほんとうに便利ですね．近々バージョンアップしてさらに高速になるということなので，楽しみに待っているところです．

★値段と性能の折り合いが……
最近，友達が「どんなパソコン買ったらいいの？」と聞いてくるのだが，どれをすすめればよいのかわからなくなっているんです．286，386，486と進歩していく中で，値段と性能の折り合いを見定めるのって，けっこう難しいものですね．さまざまな分野におけるパソコンの処理速度の比較なんてやっていただけませんか？(花見英樹)
——パソコンの処理速度というのは，単純にどうこういえないところが難しいですね．使い方によって，なにが限定要因になるかわかりませんからね．なにを勧めるかは，その人の使い方次第かな．

★最年少読者の頼もしいご意見
パソコンを始めて2年と少しの中学生です．今年1月にBASICからいきなり8086アセンブラに入りました（Cはコンパイラが嫌で．でも，アセンブラもけっこうかかるのね……)．どうも貴誌の読者というのは年齢層が高いようで，私なんか場違いの所にきたような感じです．14歳ですが，私と同じくらいの方はいらっしゃるのでしょうか．いるといいなあ．関係ないですが，私の力量の程は今やっとテキストファイルに行番号がつけられるようになった，という感じです．そんな私にとって貴誌は最高です．こんな雑誌（？）を待っていたという感じです．若輩者ですがよろしくお願いします．(大村兼)
——14歳でアセンブラに挑戦しているとは，スゴイ！ですね．私がはじめてプログラムを書いたのは，16歳のときにFORTRANでしたから．今の若者は最初からパソコンだから，ほんとうにいいよなぁ！

★Macのプログラミング雑誌が欲しい
私はMacを使用するようになってから，プログラミングをするようになりました(以前は，PC-9801を使用していましたが，ほとんどそのようなことはしませんでした)．たしかに，日本では(世界ではかな？)，MS-DOSが主流ですが，Macのプログラマも増えていると思います．プログラミング専門の雑誌もいくつかありますが，ほとんどがMS-DOSです．このへんでMacのプログラミングを扱う(Mac専門でなくてもよいのです)プログラミングの雑誌があってもよいのではないでしょうか？（藤田幸治）
——編集部でもMS-DOSマシン以外のパソコンの記事について考えてはいるのですが，ページ数の問題などもあって，なかなか思うようにいかないというのが現状です．機会があればMacも取り上げていきますので，もう少しお待ちください．

★3歳の娘とのパソコン争奪戦
いつの間にか二児の父となり，くたくたになって仕事を終わって帰宅して……．さて，パソコンでも……と思うと，いつも娘とパソコンの取り合いになる．そしていつも負けるのは私．なぜだろう？　3歳の娘に……．もう1台ほしいなー．　P.S.あべしずえさんの「コーヒーショップで」好きな歌です．(大和ひさし)
——まぁ，誰しも子供にはかなわないものですよ．娘さんに買ってあげるならMacがいいんじゃないですか．アラン・ケイの2つだか3つになる娘さんがMacを使っているフィルムを見たことがありますが，Macって女の子によく似合うんですよ．

★プログラマで先生，でも本当は音楽家志望
私は音楽家志望ですが，仕事はプログラマです．現在68000のアセンブラを使って制御系の仕事をしています．でも，これは派遣先でのこと．本社では8086の先生になって新入社員3人に教えています．とはいっても，私がこの仕事を始めて6月で1年なんです．文系大学中退の私が1年で先生だなんて，この業界はいったいなんなんでしょうか．こんな私に，もっとわかりやすい記事を！！（縣　洋志）
——たった1年で先生？　でも長さじゃないですよね．要は実力．実は音楽家志望というユニークな先生に教えられる新入社員の皆さんは幸せ者．来年は彼らが先生してたりして……．

★286でも不便はしないハズ
386だ，ウィンドウズだとさわがしいが，80286というCPUをわすれてはいないか？　現在，80286CPUを使用しているパソコンであれば，何も不便はしないハズ．なのに，なぜ386マシンやウィンドウズなど，よけいな物に手を出して，お金をかけようとするのだろう．雑誌などで大きく取り上げられると，目的もないのに買ってしまっているようだ．誇大広告の影響じゃない？　きっと使いこなせないでいるだろう．(萩原　聡)
——ハイ，ごもっともなご意見です．いくら速いマシンでも使う側によっては「猫に小判」となってしまいますよね．でも，Windowsも使ってみたい．(←コラコラ)

付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1991年9月号に掲載されたものです．

付録のフロッピーディスクは，5”2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5”や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明
ディレクトリ名：ルート
収録ファイル名：
README.BAT（ディスク内容の説明を読むためのバッチファイル，PC-9801専用）
README.DOC（ディスク内容の説明，MS-DOSテキストファイル）
ONLINE.BAT（オンラインソフトの説明を読むためのバッチファイル，PC-9801専用）
ONLINE.DOC（収録されているオンラインソフトの簡単な説明）
V.EXE（ミニページャー，PC-9801専用）

●特集「グラフィックスプログラミングの世界」
①「グラフィックスプログラミング入門」（p.2）
ディレクトリ名：TOKU
収録ファイル名：TOKU1.LZH
②「フレームバッファ活用法」（p.14）
ディレクトリ名：TOKU
収録ファイル名：TOKU2.LZH
⑤「Windowsにおけるグラフィックスプログラミングの実際」（p.47）
ディレクトリ名：TOKU
収録ファイル名：TOKU5.LZH

●プログラム「遊」工房（p.70）
「射撃」ゲーム
ディレクトリ名：PLAY
収録ファイル名：PLAY.LZH

●IC千夜一夜（p.76）
ディレクトリ名：IC
収録ファイル名：I8008.LZH（i8008アセンブラ／デバッガ）

●MS-DOSツール・ラボ（p.80）
FORTHコンパイラ「FAST」
ディレクトリ名：TOOL
収録ファイル名：TOOL.LZH

●実践C塾（p.91）
ディレクトリ名：JUKU
収録ファイル名：
JUKU.LZH
FEPC13.LZH（太田純氏作成の日本語FEP操作ライブラリ）

●綴じ込み付録
PC-9801シリーズ・グラフィックBIOS/LIO機能一覧
ディレクトリ名：DATA
収録ファイル名：GRAPH.TXT（MS-DOSテキストファイル）

●オンラインソフト採集帳（p.136）
本誌今月号のオンラインソフト採集帳で紹介したデータファイル実行ツール「えぐぜぐぜ」と，圧縮ファイルを復元するために必要な，高圧縮復元書庫管理ソフト「LHA」を収録しています．また，特集に関連するものとしてグラフィックスライブラリ「GR.LIB」も収録しました．
ディレクトリ名：ONLINE
収録ファイル名：
LHA213.EXE（LHA Ver.2.13，MS-DOS汎用）
EXEXE230.LZH（えぐぜぐぜ，MS-DOS汎用）
GRLIB078.LZH（GR.LIB，PC-9801専用）
GRDEM077.LZH（GR.LIBのデモ，PC-9801専用）

●クイズ・ランダムスクリーン（p.142）
クイズ解読ツール「PUZLE」
本誌掲載のクイズランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ替えることにより，解答を得ることができるユーティリティです．対応機種は日本電気のPC-9801シリーズです．詳しい使い方はPUZLE.DOCを参照してください．
ディレクトリ名：OMAKE
収録ファイル名：
PUZLE.EXE（解読ツール，PC-9801専用）
PUZLE.DOC（PUZLEの説明）
CONFIG.SYS（PUZLE用CONFIG.SYS）
AUTOEXEC.BAT（PUZLE用AUTOEXEC.BAT）
RNDSCREN.LZH（画像データ）
TSUBAME.LZH（8月号の解答データ）
TSUBAME.BMP（Windows 3.0用の壁紙）

※使用上の注意
本誌に掲載されたプログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．このディスクには，MS-DOSのシステムは入っていません．別途，MS-DOSが必要となります．また，ソースファイルのコンパイルなど，別途コンパイラやユーティリティが必要な場合があります．

■圧縮ファイルについて

ディスクスペースを有効に使うために，多くのファイルは圧縮した上で収録しています．「LZH」という拡張子のついたファイルは，高圧縮書庫管理ユーティリティ「LHA」によって圧縮されたファイルです．これらのファイルを利用するには，まずLHAで復元する必要があります．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

P子　「フィールド・オブ・ドリームス」っていう映画，ビデオでみたんだけど，団塊の世代向けのメルヘンっていうのが，アメリカにもあるんだなと思っちゃった．

C平　ぼくは，だいぶ前にみたよ．好感はもてるんだけど，あんまりピンとこない感じもしたんだ．

P子　日本の団塊の世代向けのメルヘンといったら「となりのトトロ」なわけじゃない．昭和30年代の田園風景のなかで，子供たちや妖怪のメルヘンが展開するわけでしょ．そういう環境で育った人にとっては，もうそれだけでなつかしいのよ．アメリカ人にとっては，あの映画がそういうなつかしさをもっているんだと思うな．ピンとこない部分ていうのは，多分そこらへんよ．原体験みたいなものが違う．

C平　うーん．鎮守の森がトウモロコシ畑で，トトロみたいな妖怪が野球選手のゴーストというわけ？

P子　そんな気がするのよ．ただ，「トトロ」の場合は，時間を逆戻りしなければ舞台がなかったけれど，「フィールド……」のほうは，現代が舞台で，現代のほうに過去がやってくるでしょう．消えていった名選手，うもれてしまった黒人運動家……．トウモロコシ畑も野球も，アメリカではあまり変化することなく続いているのよね．日本では，30年代の田園風景なんて，もう取り戻せないわけだけど．

C平　失われたものは，日本では風景であり，アメリカでは人や人の心であるということかな．

P子　ずいぶんマジじゃん．

■フィールド・オブ・ドリームス（1989・米）

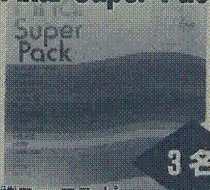
監督：フィル・アルデン・ロビンソン
出演：ケビン・コスナー

経営に苦しむ農場主が，ある日トウモロコシ畑のあいだから，不思議な声を聞く．その声に導かれるまま，農場主は野球場を作り，"何か"が起こるのを待つ．やってきたのは，往年の名野球選手……．不思議な声は，さらに"何か"を探すように暗示する．旅にでる農場主を待っていたのは……．めずらしいメルヘンタッチのアメリカ映画．

東和ビデオ，NHKエンタープライズ／ビジュアルブック　107分／15,800円

読者プレゼント

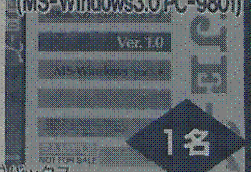
1 Final Super Pack*1



3名

提供：㈱エー・エス・ピー

2 VJE-γ (MS-Windows3.0 PC-9801)



1名

提供：㈱バックス

3 ボーランドジャパン特製カード型ラジオ

10名

提供：㈱ボーランドジャパン

4 特製Tシャツ*2

5名

提供：シエラオンラインジャパン㈱

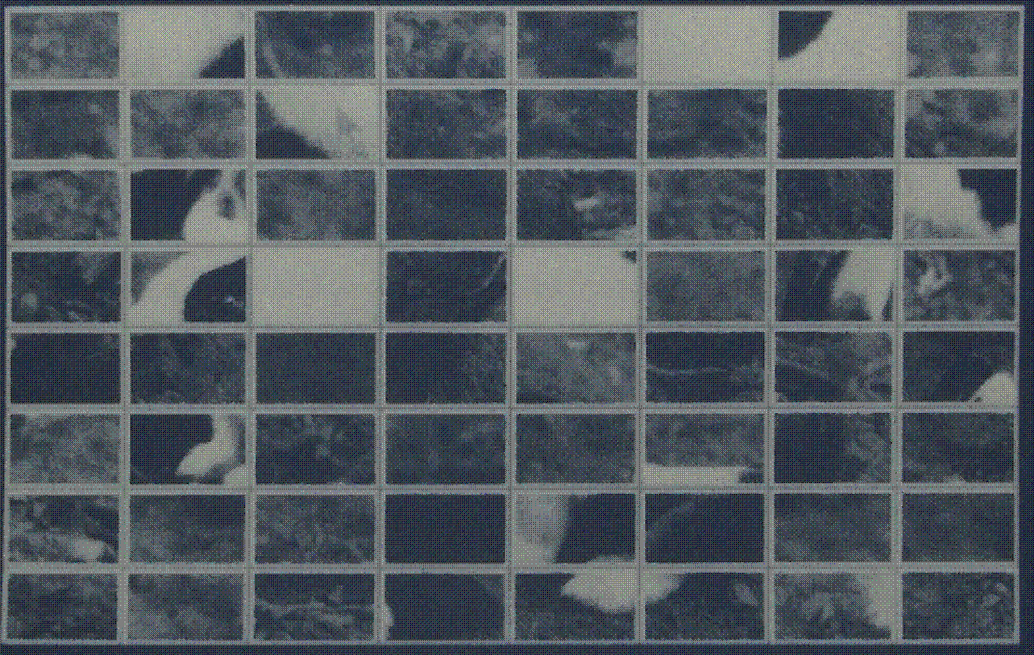
応募は本誌に添付のアンケートはがきをご利用ください．（1991年9月17日消印有効）．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

*1 Final Super Packに当選された方には，一定期間の使用の後，製品に関するレポートを提出していただきます．
*2 シエラオンラインジャパン特製Tシャツをご希望の方は，希望サイズ（S・M・L）を明記してください．

クイズ・ランダムスクリーン

**ある1枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．**

ヒント：月で餅つきをする動物



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

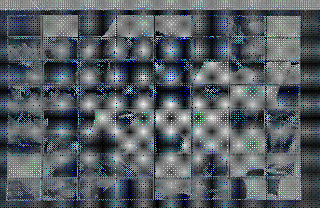
締め切り：1991年9月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

＊画像データは，付録ディスクやBBSでもサービスしています．

8月号の問題と正解

【問題】



【正解：つばめ】

月刊プログラマーズ・ページ オンラインサービス（SE-NET）のお知らせ

付録にフロッピーディスクをつけたため，ディスクによるプログラムサービスは中止いたします．オンラインサービスのBBSは，多数の読者の登録をいただいておりますので，今後も継続いたします．編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

①接続するとIDの入力を求めてきますので，GUESTと入力しリターンキーを押してください．
②お知らせを表示後，メインメニューのコマンドプロンプトが表示されます．
③月刊プログラマーズ・ページへのご意見などは「ドラゴンポスト」ボードに書き込んでください．
④ファイルの転送が終了したら，0と入力して回線を切ります．

月刊プログラマーズ・ページ バックナンバーのご案内

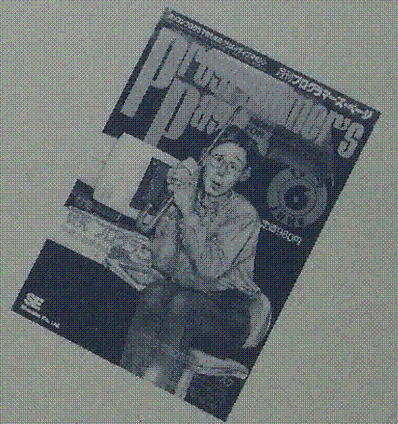
■定価・各980円

（1991年１月号「特集・プログラマの仕事場」・２月号「特集・Windowsの時代である!」・３月号「特集・この言語を使いなさい!!」・４月号「特集・Ｃ読本」は品切れです）

1991年5月号 5インチFD付き

特集・基礎力としてのアセンブラ

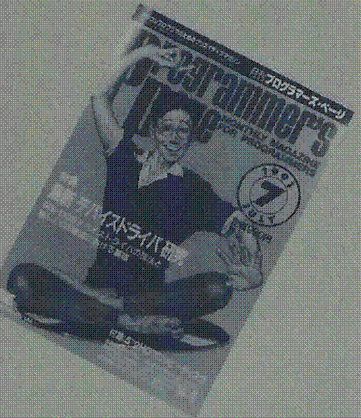
プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ．コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす．



1991年6月号 5インチFD付き

特集・研究「プログラミング統合環境」

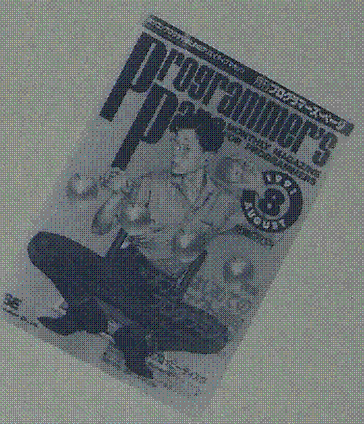
今流行りのエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する．



1991年7月号 5インチFD付き

特集・最新「デバイスドライバ」研究

パソコンの拡張性を支えるソフトウェアインターフェースであるデバイスドライバ作成の技法と特徴をMS-DOS，Windows，OS/2の各環境ごとに解説する．



1991年8月号 5インチFD付き

特集・はじめてのオブジェクト指向プログラミング

90年代のプログラミングパラダイムとして注目される“オブジェクト指向”の歴史的な発展の過程から基礎概念，分析・設計の方法までを網羅的に解説する．

「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで１か月ほどかかります）．小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を右記までお送りください．５月号の送料は１部260円です（６月号，８月号は１部81円，７月号は１部86円）．２冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください．

〒102 東京都千代田区平河町
2-4-14 平河町KSビル4階
（株）翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page
MONTHLY MAGAZINE
FOR PROGRAMMERS

月刊プログラマーズ・ページ10月号

（9月18日発売）

付録：5″ 2HDフロッピーディスク付き!!

特集
MS-DOSメモリ活用プログラミング

MS-DOSがもつ640KBのバリヤーを破るためにさまざまな試みがなされてきている．バンク式メモリ，EMS，XMSからプロテクトモードの活用まで，MS-DOSのメモリを拡張する技術を総括し，これらを利用したプログラミングについて解説を行う．

月刊プログラマーズ・ページ　1991年9月号
1991年9月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　（株）ドモン・マインズ＋佐藤裕美＋小林弘人（図版デザイン）
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

© 1991 SHOEISHA
本書の一部または全部を著作権法の定める範囲をこえ，無断で複写，転載，テープ化，データファイル化することを禁じます．
＊本書の内容に関するご質問は，ご面倒でも必ず文書にて編集部までお問い合わせください．

EDITOR'S PAGE

●「日本のビールは世界一うまい」というのは，『菊とバット』で知られるロバート・ホワイティング氏の言葉だ．確かに，ドイツのビールはこくがありすぎるかも知れないし，アメリカのビールはスカスカしている．嗜好品だから意見はいろいろあるだろうが，日本のビールが全体的にしっかりした味を出していることはまちがいない．本号が発売されるころは，多分残暑の盛り．ひときわビールが喉にしみることだろう．（TA）

●ついにというかようやくというか，タスクスイッチャを備えたMS-DOS 5.0がアメリカで発売された．おもしろいことに，DOS/VのうえにこのDOS 5.0をインストールし，$font.sys，$disp.sysなどいくつかのファイルをsetver.exeでバージョン4.00に合わせておくだけで，いとも簡単にDOS 5.0/Vとして使えるようになる．WindowsでDOSアプリケーションをスイッチングするよりもDOS 5.0のDOS-SHELLでスイッチングするほうが軽快なので，Windowsアプリケーションを使用しないのであればこのほうが使い勝手がいいようだ．（嘉平）

●朝起きたらものすごく体が痛かったので，きっと寝違えたんだろうな，とタカをくくっていたら，いつまでたっても良くならない．これは重症かもしれない！　とあわてて病院へ行ったら，なんと……神経痛だって．これには情けなさに思わず笑ってしまいました．寝冷えが原因ではないかとのことですが，皆さんもご注意を．（えっ！　そんな歳じゃないって，私だってそんな歳じゃありませんよ）（ま）

●プログラマの仕事場で取材したアンテナハウスは，なんとプログラマーズ・ページ編集部から徒歩5分，同じビルにあるお弁当屋さんには，よくお世話になっていただけにびっくりでした．「私の顔なんか見たい人はいませんよ」と最後まで写真を撮らせてくださらなかった小林社長はとても親切な方で，わざわざ出張先から校正を送ってくださいました．「なかなかよくまとまっていますね」の一言に，私は作文の添削を受けた生徒の気分，思わず顔が赤くなってしまいました．（kos）

●先月号の後記にフッとバイクが欲しいなって書いたおかげで，スズキのGSX250S-KATANAを「衝動買い」してしまった．とっとと梅雨が開けることを心待ちにしながら，納車以来，毎週末は300キロから400キロの中距離ツーリングを取り合えず満喫しています．でも車を売ってバイクを買うなんて，まさに東京の土地事情を如実に表しているような気もする．でも冬になれば，また車買うんだろうな……こんなことだから貧乏な生活から抜け出せないのかもしれない．

　ところで，先日読者ハガキを読んでいたら「よく利用するソフトウェア」の項目に『食添くん』ってのがあったのだが，いったいどんなソフトウェアで，また何と読めばいいのだろうかと気になっている．知っている人がいたら御一報願いたい．（Yoshi）

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
- □1. 書店店頭　□2.他社出版物(　　　)
- □3.小社出版物　□4. DM　□5.口コミ

パソコンの所有形態(機種名)
- □1. 個人所有　(　　　)
- □2. 職場にある(　　　)
- □3. 持っていない

本誌を購入した書店　　市(区)　　書店

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

希望項目 ＼ 広告資料請求No.												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

広告
'ER-
と思
社前
くや
しま
係な
です。
格の
結局、
ば自
ょう
違い
す。
がで
なる
性格
来た
と決
では
いる
がク
ある、
は困

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

郵便はがき

料金受取人払

麹町局承認

3908

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ
READER'S SURVEY CARD　愛読者カード

氏名	性別 □1男 □2女	生年月日 19　年　月　日
住所 〒		電話 (　　) －

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（　　　　）

ユーザー区分
□1 プロ
□2 マニア
□3 一般ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

料金受取人払

麹町局承認

3910

差出有効期間
平成4年11月
30日まで

切手不要

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.	

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓　　名	5	生年月日	19　年　月　日

6	自宅住所	〒　電話（　　）
7	会社住所	〒　電話（　　）

資料送付希望先の番号に○印をつけてください。

払込票
通常払込料金加入者負担
口座番号 東京 4-154493
加入者名 株式会社 翔泳社
金額 11760
払込人住所氏名 ※
受付局日附印
備考
切り取らないで郵便局にお出しください。
記載事項を訂正した場合は、その箇所に訂正印を押してください。

払込通知票
通常払込料金加入者負担
口座番号 東京 4-154493
金額 11760
加入者名 株式会社 翔泳社
料金 特殊 円
払込人住所氏名 ※(郵便番号) TEL( )
受付局日附印
各票の※印欄は、払込人において記載してください。
この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたり しないでください。 (郵政省)

イフボート

Xソフトウェアから
こ代表される教育・学習用
開発のための数多くのシス
発している会社だ。現在、
独自のリサーチ能力とマー
、例えばDOS/16Mな
こも高く評価されているシ
ている。ユーザー層は広範
屈や時代をリードする企画
ンがいるほどの定評を誇る。
用に対する考え方について、
スの田先社長を訪れて聞い

い道具とより専門的な部品の提供に当たっています。ですからプログラマーズ・ページの読者とうちのターゲットはオーバーラップするところがありますね。」

設立から現在まで、会社が発展するきっかけとなった商品は？

「まず'81年の設立当初は、「CP/M」というOSのソフトを販売しました。つぎにMS-DOSの時代になり、そのときに出したのが『Lattice C コンパイラ』です。この製品によって会社は成長しましたが、ある面では『Lattice C コ

「これはいろいろな説があるんですけど(笑)、ま、今日のところはヒッチコックの映画から取ったということにしておきましょう。」

頭のいい人を求めています

貴社では、今までどのようにして人材募集を行ってきましたか？ また現在は？

「やはり中途採用が多いですね。新規採用者もいますが、こちら側から大学や専門学校等に新卒者の募集を働きか

ことは全くありません。募集広告いて言えば、自社媒体の「PER-CTIVE」に優るものはないと思す。また、ライフボートを入社前知っていた人は入社後もうまくやいるように思えます。」

する際に技術力や実績は考慮しま？

の人のキャリアとかは特に関係なすね。頭のいい人材が欲しいです。能力よりモラルの高い人や性格の人が望まれますね。ですが、結局、いい人であれば仕事を任せれば自置かれた立場も理解するでしょうし、大事な商談の時も丁寧な言葉遣いで対処することができると思います。またこのように、その時々の分析ができる人であれば性格も自然とよくなるでしょう。ですから頭はいいのに性格が悪いと言われている人はうちに来たらよいと思います(笑)。

ただ困るのは、自分は頭がいいと決め込んでいる人ですね。特に日本では技術者がものすごく甘やかされているんじゃないですかね。技術者だけがクリエイティブで高いポジションにある、だから自分も特別だと考えられては困

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。　（郵政省）

郵便はがき

1 0 2 - 0 0

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

綴じ込みの

□10文系学生　□11その他(　　　　)　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話　(　　)	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話　(　　)	

人材募集
広告企画

ちょっと気になるこの会社

株式会社ライフボート

ライフボートは、UNIXソフトウェアからAdvanced RUNシリーズに代表される教育・学習用製品に至るまで、ソフトウェア開発のための数多くのシステム・ソフトウェアを企画・開発している会社だ。現在、社員が約30名と小さいながら、独自のリサーチ能力とマーケティング能力、技術力を生かし、例えばDOS／16Mなど大手コンピュータ・メーカーにも高く評価されているシステム・ソフトウェアを提供している。ユーザー層は広範囲に及んでおり、最新技術の発掘や時代をリードする企画力にかけてはライフボートファンがいるほどの定評を誇る。そんな同社の事業内容、人材採用に対する考え方について、編集部は神田にある本社オフィスの田先社長を訪れて聞いてみた。

システムソフトウェアを幅広く

ライフボートでは数々のパッケージソフトを提供していますが、会社の特徴を教えてください。

「うちの会社は、簡単に言うとシステムソフトに特化した技術型商社です。プログラミングツールとソフトウェアのコンポーネントでシステムソフトウェアの問題点を解決する、つまり非常にプロフェッショナルなプログラマから準プログラマまでを対象に、使い易い道具とより専門的な部品の提供に当たっています。ですからプログラマーズ・ページの読者とうちのターゲットはオーバーラップするところがありますね。」

設立から現在まで、会社が発展するきっかけとなった商品は？

「まず'81年の設立当初は、「CP/M」というOSのソフトを販売しました。つぎにMS-DOSの時代になり、そのときに出したのが『Lattice Cコンパイラ』です。この製品によって会社は成長しましたが、ある面では『Lattice Cコンパイラ』の下にライフボートという会社が隠れたような気がします。ですから、その後はひとつの製品が全製品の売上の20%を超えないように工夫しています。

最近では、システムソフトウェアを軸に、上は『VAX/VMS』のユーティリティから、下は『RUNシリーズ』まで幅広くラインナップを揃え、そのひとつひとつの芽を着実に伸ばしていこうと考えています。」

ライフボートという会社名は、数あるソフトハウスの中でもユニークだと思いますが、この名称の由来は？

「これはいろいろな説があるんですけど(笑)、ま、今日のところはヒッチコックの映画から取ったということにしておきましょう。」

頭のいい人を求めています

貴社では、今までどのようにして人材募集を行ってきましたか？ また現在は？

「やはり中途採用が多いですね。新規採用者もいますが、こちら側から大学や専門学校等に新卒者の募集を働きかけることは全くありません。募集広告について言えば、自社媒体の「PERSPECTIVE」に優るものはないと思います。また、ライフボートを入社前から知っていた人は入社後もうまくやっているように思えます。」

採用する際に技術力や実績は考慮しますか？

「その人のキリャアとかは特に関係ないですね。頭のいい人材が欲しいです。よく能力よりモラルの高い人や性格のいい人が望まれますね。ですが、結局、頭のいい人であれば仕事を任せれば自分の置かれた立場も理解するでしょうし、大事な商談の時も丁寧な言葉遣いで対処することができると思います。またこのように、その時々の分析ができる人であれば性格も自然とよくなるでしょう。ですから頭はいいのに性格が悪いと言われている人はうちに来たらよいと思います(笑)。

ただ困るのは、自分は頭がいいと決め込んでいる人ですね。特に日本では技術者がものすごく甘やかされているんじゃないですかね。技術者だけがクリエイティブで高いポジションにある、だから自分も特別だと考えられては困

るわけです。ですから私は、そういう人とか抽象化能力のないソフトウェア技術者に対しては批判的です。もちろん、うちの社員全員が頭がいいとは言わないですけど。」

柔軟性のない人に対してはどうですか？

「特に頭が硬直化してしまっている人は例外ですが、単にC言語が分からない、英語ができない、ハードウェアが苦手、赤面症で接待ができないというような人でも、うちに来れば直りますよ。」

具体的にはどのような社員教育をしていますか？

「例えば、英語を覚えさせるには一人で外国に出すとかね。泳ぎのできない人には、一度プールに突き落として必死になって浮き上がってから手足の動かし方を教えるといいと言いますが、それと同様に一度放り出してみるわけです。そこで浮き上がって来ない人もいますが(笑)、浮き上がって何かができるようになったという社員がほとんどですね。また、外部のセミナーには積極的に社員を出している方だと思います。今後は教育体制をもっとシステマティックにしてうちならではのメソッドを確立する必要があると思っています。」

社員の仲のよいのが悩み

社員の待遇面はどうですか？

「年功序列とか男女差別は否定しますが、例えば先輩を敬うとか女性にドアをひいてあげるとかまで否定はしません。ただ、単に年を取ったというだけで管理職にしても、その人に管理能力がなかったらその人自身も部下も不幸になりますからね。じっくり考えないといけない問題ですよね。

社員の収入には二つあります。一つは純粋な労働の対価としての賃金。もう一つは利益が出たときに還元する利益配分。この二つはそれぞれ独立したものです。ですから賃金は能力や仕事の難度によってメリハリを付けて支払う一方、利益配分は、例えば、仕事はそれほどできなくても会社に運を呼ぶ人(笑)などに多く還元するとかダイナミックに決断しています。」

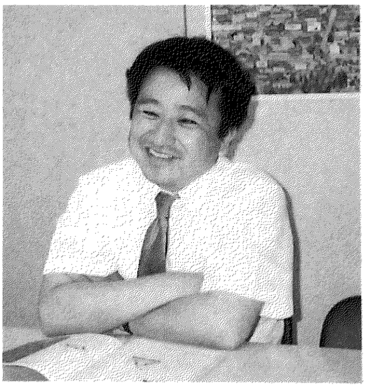
給与例を拝見させていただくとかなり優遇されていると思うのですが？　残業もあまり多くないようですし。

「いえ、逆にそれぐらいは取ってもらわなきゃ困ります。また、残業をすべて認めるわけでなく残業しなくてもいいようにコントロールしてもらわないと。だから残業するよりも早く帰って夢の中で仕事のことを考えてもらいたいですね。」

社内でのクラブ活動ですとか飲み会のようなものは盛んですか？

「どうでしょうか？　僕の考えなんですが、社長が飲みニュケーションを大事にしているという理由で社員にビールをつがせたりする会社は疑問ですね。今の時代、仕事以外に趣味とかでみんな忙しいでしょう。互いのプライバシーはすごく大事だと思いますからそれは尊重しないと。だから、僕は最近うちの社員の仲がみんな良すぎるのではないかと真剣に悩んでいます。仕事上はもっと冷たい関係が望まれますね。」

回りくどい表現が好まれる日本の建前社会。そのなかで田先社長は自社への適性をズバリ「頭のいい人」と語った。求人に対する媚びが全く感じられない発言には潔いものがある。確かに若干30人の社員で10億円以上もの年商を生むのだから、社員には本物の実力が要求されるだろう。また、仕事以外にもコミュニケーションを強要される日本の会社にうんざりしている人には、仕事に専念できるこの会社の環境は好ましいと言えよう。コンピュータ・フリークとして鋭いセンサーで時代に切り込む自信のある人は、一度、田先社長に会ってみるのも面白いのではないだろうか。



〈プロフィール〉
田先政秀
たさきまさひで

1951年、新潟生まれ。父はサーカス付き獣医で、幼少時から高校まで全国を転々とする。名古屋の高校を中退。その後数々の会社を経て、22歳のときに開発プログラマとなり、30歳でライフボートを設立、代表取締役として現在に至る。

第三話 最終回
明日に向かって笑え！の巻

川辺義宏クンは、数理技研をどう考えているか。

㈱数理技研は、今回大々的に、
強力なスタッフを多数募集します。
これは、そのための、
人材募集シリーズ広告・第3弾です。
興味をもたれた方の
ご応募をお待ちしています。

僕の物語も、今月号で最終回となってしまった。
ネットワークソフトの開発、X-Windowsのサーバの移植、
そして現在はCD-ROMとUNIX機の接続と、順調に僕は技術者の道を歩んできたわけだが、
どうも根が真面目なため堅い話ばかりしてしまったようだ。
数理技研は仕事もマジだが、遊びにもマジである。
まず、プロジェクトの打ち上げと称する飲み会がよく行われる。
これは、新宿という魅惑的な場所柄に加えて、
予算に「プロジェクト打ち上げ費用」が組み込まれていることが拍車をかけている。
昨年などは、大胆にも社員全員でタイに研修旅行へ行ってしまった。
（これが何の研修だったかを、僕は思い出すことができない。）
小さい会社ながら、国際感覚はなかなかのものがある。
海外に取引先があるのはもちろんのこと、中国からの研修生も受け入れている。
また、国内・国外を問わずセミナーには積極的に参加し、
海外ネットにも加入しているので、社内には常に最新の情報がある。
最後に、プログラマーズ・ページの読者にひとこと。
この雑誌を面白いと思って読んでいる君なら、この会社はもっと面白いと感じるはずだ。
「ワッハッハ」と毎日思うことのできる自分を、僕は幸せ者だと思う。
（完）

DATA

■募集要項

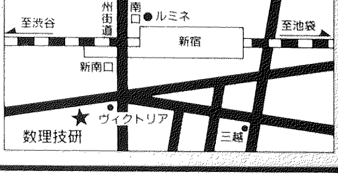
職種●システムエンジニア
資格●大卒以上もしくは大卒程度の学力を有する28才位までの男女。
※やる気・好奇心があり、技術を吸収する力がある方。
給与●経験・年齢を考慮の上、当社規定により優遇します。（'91年新大卒基本給18万3千円）
待遇●昇給年1回（'91年度実績平均8.0%）、賞与年2回（'90年度実績平均7.2ヵ月）、各種社会保険完備、交通費全額支給、諸手当（子供・役職）、制度（住宅購入資金融資・退職金）、施設（独身寮・リゾートマンション）
休日休暇●完全週休2日制（土日）、祝日、年末年始、夏期休暇
勤務地●東京および山梨（'92年4月開設）
勤務時間●フレックスタイム制
教育制度●数理アカデミアと称する体系的な理論学習の講座制度あり。技術セミナーへの参加や論文発表の機会多数。
応募●履歴書をご郵送下さい。追って詳細をご連絡します。
交通●JR新宿駅南口下車徒歩2分

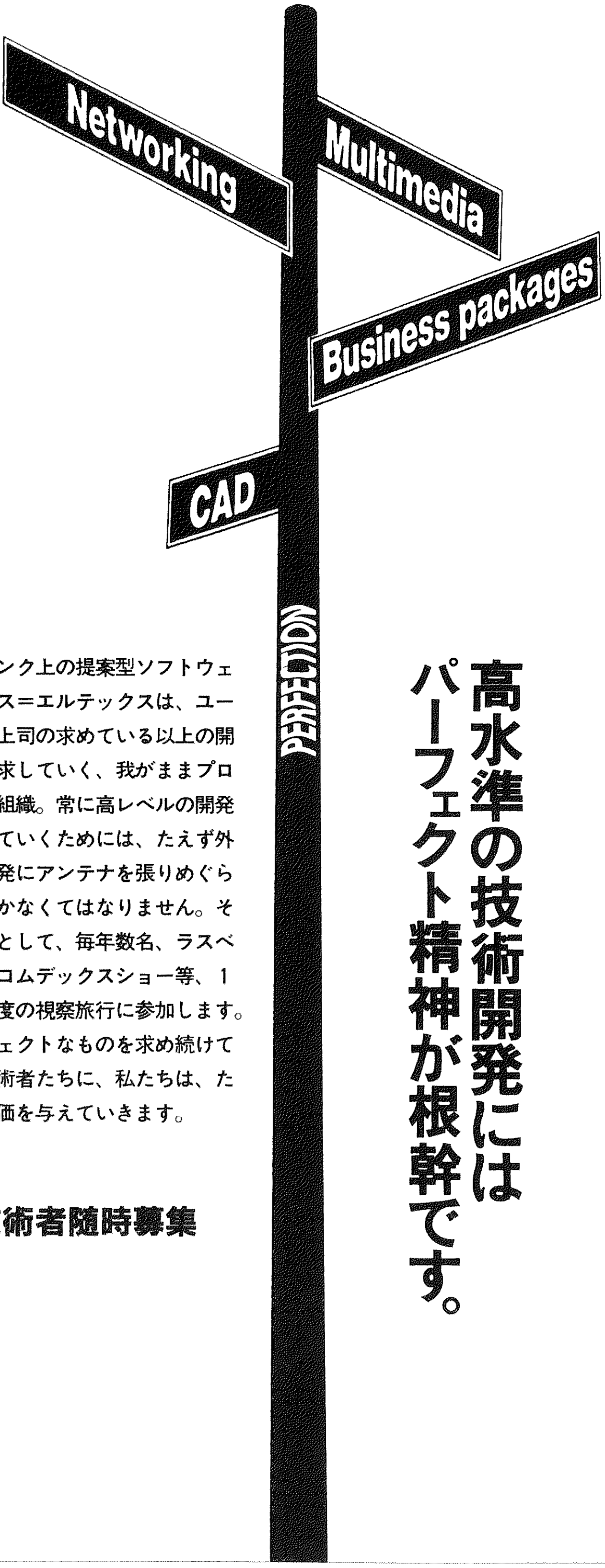
■会社概要

設立●1979年3月
資本金●4800万円
売上高●11億円（'90年8月期）、13億円（'91年8月期見込み）
社員数●60名（男45名、女15名）
事業内容●①UNIXはじめOS・通信系基本ソフトの開発
②大規模・分散処理システムの開発と同インテグレーション
③上記①、②の研究開発
④宇宙開発・核融合・数値解析など、科学技術計算の先端分野での研究開発
特徴●企業というより技術研究機関
促成栽培を否定した技術者教育
開発環境の完備
タイムカードなし、服装自由など自主性尊重
事業所●本社：東京都新宿
研究所：山梨県竜王
※'92年4月、山梨県竜王町のソフトウェアパーク内に山梨研究開発センターを開設

主要取引先●宇宙科学研究所、日本原子力研究所、NTT、NTTデータ通信、地球科学総合研究所、日本IBM、富士通、富士通研究所、日本電気、本田技研、日産自動車、IHI、住友電工、日立製作所、東芝、ソニー、アンリツ、日本信号、日本無線、ファナック、ヤマハ、NKKなど
技術提携先●UNIXインターナショナル、中国科学院軟件研究所、MENTAT Inc.（米）

株式会社 数理技研
〒160 東京都新宿区新宿4-1-9 新宿ユースビル
TEL 03-3356-2781㈹





高水準の技術開発にはパーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

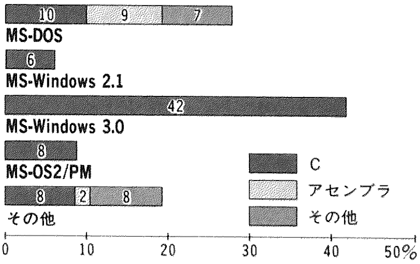
会社概要　ELTEX

設　立	昭和60年12月14日
資本金	1,000万円
事業内容	パソコンを中心としたシステムの提案・開発・販売　①CD-ROM・LD など新しいメディアに対応したシステムの開発　②ネットワークの端末システムの開発　③特定業種向けCADの開発・販売　④基本ソフトの開発
特　徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している。また、ＣＡＤを中心とした自社プロダクツにも積極的で、その比率も増加している。基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、ＯＥＭ製品のＢＩＯＳ・ユーティリティなどを企画から手掛けている。
取引先	三菱電機、三菱自動車、日本航空　他
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募集要項　ELTEX

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上　20～36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心･向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）～46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理１種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理１種・住宅手当含む）
手　当	時間外手当、役職手当（5000円～６万円）、住宅手当（１万5000円）、資格手当（２種：5000～特種：２万円）、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月）、決算賞与年１回（平成１年度実績１ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00～16:00）試用期間中は標準時間9:00～17:45
休日休暇	完全週休２日制（土・日）、祝日、年３回連休（GW･夏期･年末年始）、有給･特別･休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書(写真貼付)、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045)322-9667　総務課

最近のOS・言語別開発動向



採用HOT・LINE
TEL.(045)322-9667

株式会社エルテックス

PC-9801シリーズ・グラフィックBIOS／LIO機能一覧

(W)(4)などはサイズを表わします(W：ワード,4：4バイト)
明記していないものは1バイト

●グラフィックBIOS(INT 18H)

グラフィック画面の表示開始
AH←40H

グラフィック画面の表示停止
AH←41H

表示領域の設定
AH←42H
CH←ディスプレイモードの指定
b7,b6 ←VRAM領域(01:Upper／10:Lower／11:All)
b5 ←ディスプレイモード(0:カラー／1:モノクロ)
b4 ←表示バンク(0:バンク0／1:バンク1)

パレットレジスタの設定
AH←43H
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
[BX+04H] パレットレジスタの設定(4)
+00H b6〜b4:パレットNO6(b6:G b5:R b4:B)
b2〜b0:パレットNO7(b2:G b1:R b0:B)
+01H b6〜b4:パレットNO4(b6:G b5:R b4:B)
b2〜b0:パレットNO5(b2:G b1:R b0:B)
+02H b6〜b4:パレットNO2(b6:G b5:R b4:B)
b2〜b0:パレットNO3(b2:G b1:R b0:B)
+03H b6〜b4:パレットNO0(b6:G b5:R b4:B)
b2〜b0:パレットNO1(b2:G b1:R b0:B)

ボーダーカラーの設定
AH←44H
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
[BX+01H]←セットするボーダーカラーコード
b6:G b5:R b4:B

ドットの書き込み
AH←45H
CH←描画画面の指定
b7 ←解像度モード(0:標準解像度/1:専用高解像度)
b6 ←描画画面範囲(0:Lower,All/1:Upper)
b5,b4 ←描画画面ナンバー(00:P1(P4)/01:P2(P5)/10:P3(P6)/11:P1,P2,P3(P4,P5,P6))
ES←描画パターンバッファのセグメントアドレス
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←3画面同時書き込み描画オペレーションモード指定
b2:描画画面P3(P6)←0:Reset/1:Set
b1:描画画面P2(P5)←0:Reset/1:Set
b0:描画画面P1(P4)←0:Reset/1:Set
[BX+02H]←単一画面描画オペレーションモード指定
b1,b0←00:Replace/01:Complement/10:Clear/11:Set
[BX+08H]←描画開始アドレスX座標(W)
[BX+0AH]←描画開始アドレスY座標(W)
[BX+0CH]←描画画面長ドット数(W)
[BX+0EH]←描画パターンバッファ開始オフセットアドレス(W)

ドットの読み出し
AH←46H
CH←描画画面の指定
b7 ←解像度モード(0:標準解像度/1:専用高解像度)
b6 ←描画画面範囲(0:Lower,All/1:Upper)
b5,b4 ←描画画面ナンバー(00:P1/01:P2/10:P3/11:P1,P2,P3)
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
ES←読み込みバッファ(1〜3)のセグメントアドレス
[BX+08H]←描画開始アドレスX座標(W)
[BX+0AH]←描画開始アドレスY座標(W)
[BX+0CH]←描画画面長ドット数(W)
[BX+10H]←読み込みバッファ1の開始オフセットアドレス(W)
[BX+12H]←読み込みバッファ2の開始オフセットアドレス(W)
[BX+14H]←読み込みバッファ3の開始オフセットアドレス(W)

直線・矩形の描画
AH←47H
CH←描画画面の指定
b7 ←解像度モード(0:標準解像度/1:専用高解像度)
b6 ←描画画面範囲(0:Lower,All/1:Upper)
b5,b4 ←描画画面ナンバー(00:P1(P4)/01:P2(P5)/10:P3(P6)/11:P1,P2,P3(P4,P5,P6))
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←3画面同時書き込み描画オペレーションモード指定
b2:描画画面P3(P6)←0:Reset/1:Set
b1:描画画面P2(P5)←0:Reset/1:Set
b0:描画画面P1(P4)←0:Reset/1:Set
[BX+02H]←単一画面描画オペレーションモード指定
b1,b0←00:Replace/01:Complement/10:Clear/11:Set
[BX+03H]←描画開始方向の指定
00H:下/01H:右下/02H:右/03H:右上/04H:上/05H:左上/06H:左/07H:左下
[BX+08H]←描画開始アドレスX座標(W)
[BX+0AH]←描画開始アドレスY座標(W)
[BX+16H]←描画終了アドレスX座標(W)
[BX+18H]←描画終了アドレスY座標(W)
[BX+20H]←線種パターン(2)
b0〜b15←描画パターン
[BX+28H]←描画タイプ
01H:直線/02H:矩形/04H:円弧

円弧の描画
AH←48H
CH←描画画面の指定
b7 ←解像度モード(0:標準解像度/1:専用高解像度)
b6 ←描画画面範囲(0:Lower,All/1:Upper)
b5,b4 ←描画画面ナンバー(00:P1(P4)/01:P2(P5)/10:P3(P6)/11:P1,P2,P3(P4,P5,P6))
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←3画面同時書き込み描画オペレーションモード指定
b2:描画画面P3(P6)←0:Reset/1:Set
b1:描画画面P2(P5)←0:Reset/1:Set
b0:描画画面P1(P4)←0:Reset/1:Set
[BX+02H]←単一画面描画オペレーションモード指定
b1,b0←00:Replace/01:Complement/10:Clear/11:Set
[BX+03H]←描画開始方向の指定(1)
00H:下/01H:右下/02H:右/03H:右上/04H:上/05H:左上/06H:左/07H:左下
[BX+08H]←描画開始アドレスX座標(W)
[BX+0AH]←描画開始アドレスY座標(W)
[BX+0CH]←描画総ドット数(W)
[BX+1AH]←マスキングドット数(W)
[BX+1CH]←半径(W)
[BX+20H]←線種パターン(2)
b0〜b15←描画パターン
[BX+28H]←描画タイプ
01H:直線/02H:矩形/04H:円弧(つねに04Hを指定)

グラフィック文字の描画
AH←49H
CH←描画画面の指定
b7 ←解像度モード(0:標準解像度/1:専用高解像度)
b6 ←描画画面範囲(0:Lower,All/1:Upper)
b5,b4 ←描画画面ナンバー(00:P1(P4)/01:P2(P5)/10:P3(P6)/11:P1,P2,P3(P4,P5,P6))
DS←パラメータリスト先頭のセグメントアドレス
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←3画面同時書き込み描画オペレーションモード指定
b2:描画画面P3(P6)←0:Reset/1:Set
b1:描画画面P2(P5)←0:Reset/1:Set
b0:描画画面P1(P4)←0:Reset/1:Set
[BX+02H]←単一画面描画オペレーションモード指定
b1,b0←00:Replace/01:Complement/10:Clear/11:Set
[BX+03H]←描画開始方向の指定
00H:下/01H:右下/02H:右/03H:右上/04H:上/05H:左上/06H:左/07H:左下
[BX+08H]←描画開始アドレスX座標(W)
[BX+0AH]←描画開始アドレスY座標(W)
[BX+0CH]←第1描画方向ドット数(W)
[BX+1EH]←第2描画方向ドット数(W)
[BX+20H]←基本パターン情報(8)

	b0	b1	b2	b3	b4	b5	b6	b7
+00H	x	x	x	x	x	x	x	x
+01H	x	x	x	x	x	x	x	x
+02H	x	x	x	x	x	x	x	x
+03H	x	x	x	x	x	x	x	x
+04H	x	x	x	x	x	x	x	x
+05H	x	x	x	x	x	x	x	x
+06H	x	x	x	x	x	x	x	x
+07H	x	x	x	x	x	x	x	x

描画モードの設定
AH←4AH
CH←描画タイミングモードの設定
b7〜b5←0
b4 ←0:フラッシュ描画/1:フラッシュレス描画
b3〜b0←0110

●グラフィックLIO

グラフLIOの初期化(INT A0H)
(呼)DS ←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
(返)AH←終了条件(00H:正常終了)

モード設定(INT A1H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←画面モード
00H:カラーグラフィックモード
01H:グラフィックモード
02H:専用高解像度ディスプレイモード
03H:専用高解像度カラーディスプレイモード
FFH:今までのモード
[BX+01H]←画面スイッチ
00H:グラフィック表示あり
01H:グラフィック表示あり(高速書き込み)
02H:グラフィック表示なし
03H:グラフィック表示なし(高速書き込み)
FFH:今までのモード
[BX+02H]←アクティブ画面
00H〜0BH:アクティブ画面の番号
FFH:画面モードの変更がない場合は,現在のアクティブ画面を引き継ぐ.画面モードの変更があった場合はアクティブ画面を0とする.
[BX+03H]←ディスプレイ画面
00H〜1FH:ディスプレイ画面の番号
FFH:画面モードの変更がない場合は,現在のディスプレイ画面を引き継ぐ.画面モードの変更があった場合はディスプレイ画面を1とする.
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し)

描画領域の指定(INT A2H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←ビューポートの左上X座標(W)
[BX+02H]←ビューポートの左上Y座標(W)
[BX+04H]←ビューポートの右下X座標(W)
[BX+06H]←ビューポートの右下Y座標(W)
[BX+08H]←ビューポート内を塗りつぶす色
00H〜07H(0FH):指定パレット番号
FFH:塗りつぶしを行わない
[BX+09H]←ビューポートの外枠を描く色
00H〜07H(0FH):指定パレット番号
FFH:外枠を描かない
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し)

フォアグラウンドカラー・バックグラウンドカラーの指定(INT A3H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←未使用
[BX+01H]←バックグラウンドカラー
00H〜07H(0FH):指定パレット番号
FFH:今までの色を引き継ぐ
[BX+02H]←ボーダーカラー
00H〜07H(0FH):表示色コード
FFH:今までの色を引き継ぐ
[BX+03H]←フォアグラウンドカラー
00H〜07H(0FH):指定パレット番号
FFH:今までの色を引き継ぐ
[BX+04H]←パレットモード
00H: 8 色中 8 色
01H:4096色中 8 色
02H:4096色中16色
(返)AH←終了条件(00H:正常終了)

パレット番号と表示色の対応(INT A4H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
8 色中 8 色モードの場合
[BX+00H]←パレット番号
00H〜07H
[BX+01H]←表示色コード
00H〜07H(b2:G b1:R b0:B)
4096色中 8 色モード
[BX+00H]←パレット番号
00H〜07H
[BX+01H]←表示色コード(2)
0000H〜0FFFH(b11〜b8:G b7〜b4:R b3〜b0:B)
4096色中16色モード
[BX+00H]←パレット番号
00H〜0FH
[BX+01H]←表示色コード(2)
0000H〜0FFFH(b11〜b8:G b7〜b4:R b3〜b0:B)
(返)AH←終了条件(00H:正常終了)

描画領域のクリア(INT A5H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
(返)AH←終了条件(00H:正常終了)

ドットの書き込み(INT A6H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]← X 座標(W)
[BX+02H]← Y 座標(W)
[BX+04H]←パレット番号
00H〜07H(0FH):指定パレット番号
FFH:動作モードにしたがった表示色
(返)AH←終了条件(00H:正常終了)

直線・矩形の描画(INT A7H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←描画開始点 X 座標(W)
[BX+02H]←描画開始点 Y 座標(W)
[BX+04H]←描画終了点 X 座標(W)
[BX+06H]←描画終了点 Y 座標(W)
[BX+08H]←パレット番号
00H〜07H(0FH):指定パレット番号
FFH:フォアグラウンドカラーで描画
[BX+09H]←描画コード
00H:直線描画指定/01H:矩形描画指定/02H:矩形を塗りつぶす描画指定
[BX+10H]←ラインスタイルスイッチ
00H:ラインスタイル指定なし/01H:ラインスタイル指定あり
[BX+11H]←ラインスタイル(W)
b15〜b0:ドットパターン
(返)AH←終了条件(00H:正常終了)

円・楕円の描画(INT A8H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←中心点 X 座標(W)
[BX+02H]←中心点 Y 座標(W)
[BX+04H]← X 方向半径(W)
[BX+06H]← Y 方向半径(W)
[BX+08H]←パレット番号
00H〜07H(0FH):指定パレット番号
FFH:フォアグラウンドカラーで描画
[BX+09H]←各フラグ
b0:開始点指示の有無(0:なし／1:あり)
b1:開始線分指示の有無(0:なし／1:あり)
b2:終了点指示の有無(0:なし／1:あり)
b3:終了線分指示の有無(0:なし／1:あり)
b4:開始点・終了点一致時の描画方法の指定(0:全楕円/1:一致点のみ)
b5:塗りつぶし指示の有無(0:なし／1:塗りつぶす)
b6:タイルパターン指示の有無(0:なし／1:あり)
b7:未使用
[BX+10H]←開始点 X 座標(W)
[BX+12H]←開始点 Y 座標(W)
[BX+14H]←終了点 X 座標(W)
[BX+16H]←終了点 Y 座標(W)
[BX+18H]←タイルパターンサイズまたは塗りつぶし色のパレット番号
[BX+19H]←タイルパターン領域のオフセットアドレス(W)
[BX+21H]←タイルパターン領域のセグメントアドレス(W)
(返)AH←終了条件(00H:正常終了 06H:演算オーバーフロー)

指定色による塗りつぶし(INT A9H)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←開始点 X 座標(W)
[BX+02H]←開始点 Y 座標(W)
[BX+04H]←領域色パレット番号
00H〜07H(0FH):指定パレット番号
FFH:フォアグラウンドカラーで塗りつぶす
[BX+05H]←境界色パレット番号
00H〜07H(0FH):指定パレット番号
FFH:領域色と同じ色を境界色にする
[BX+06H]←ワーク域最終オフセット
[BX+08H]←ワーク域先頭オフセット
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し 07H:ワーク域不足で処理中断)

タイルパターンによる塗りつぶし(INT AAH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←開始点 X 座標(W)
[BX+02H]←開始点 Y 座標(W)
[BX+04H]←未使用
[BX+05H]←タイルパターンの大きさ
モノクロ:01H〜FFH
カラー:03H〜FFH
[BX+06H]←タイルパターンオフセットアドレス(W)
[BX+08H]←タイルパターンセグメントアドレス(W)
[BX+10H]←境界表示色
00H〜07H(0FH):指定パレット番号
FFH:領域色と同じ色を境界色にする
[BX+11H]←未使用(5)
[BX+16H]←ワーク域最終オフセットアドレス
[BX+18H]←ワーク域先頭オフセットアドレス
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し 07H:ワーク域不足で処理中断)

画面イメージの格納(INT ABH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←指定領域左上点 X 座標(W)
[BX+02H]←指定領域左上点 Y 座標(W)
[BX+04H]←指定領域右下点 X 座標(W)
[BX+06H]←指定領域右下点 Y 座標(W)
[BX+08H]←格納域オフセットアドレス(W)
[BX+10H]←格納域セグメントアドレス(W)
[BX+12H]←格納域のサイズ(W)
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し)

画面イメージの復帰(INT ACH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←描画領域左上点 X 座標(W)
[BX+02H]←描画領域左上点 Y 座標(W)
[BX+04H]←格納域オフセットアドレス(W)
[BX+06H]←格納域セグメントアドレス(W)
[BX+08H]←格納域のサイズ(W)
[BX+10H]←描画モード
00H:通常/01H:反転/02H:論理和/03H:論理積/04H:排他的論理和
[BX+11H]←カラースイッチ
00H:フォアグラウンド・バックグラウンドカラーの指定なし,格納領域内描画情報の画面モードは現在のカラーモードとして扱う
01H:フォアグラウンド・バックグラウンドカラーの指定あり,格納領域内描画情報の画面モードはモノクロモードとして扱う
[BX+12H]←フォアグラウンドカラー
00H〜07H(0FH):指定パレット番号
[BX+13H]←バックグラウンドカラー
00H〜07H(0FH):指定パレット番号
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し)

日本語の描画(INT ADH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←描画領域左上点 X 座標(W)
[BX+02H]←描画領域左上点 Y 座標(W)
[BX+04H]←表示する文字のJISコード(W)
[BX+06H]←描画モード
00H:通常/01H:反転/02H:論理和/03H:論理積/04H:排他的論理和
[BX+07H]←カラースイッチ
00H:フォアグラウンド・バックグラウンドカラーの指定なし,格納領域内描画情報の画面モードは現在のカラーモードとして扱う
01H:フォアグラウンド・バックグラウンドカラーの指定あり,格納領域内描画情報の画面モードはモノクロモードとして扱う
[BX+08H]←フォアグラウンドカラー
00H〜07H(0FH):指定パレット番号
[BX+09H]←バックグラウンドカラー
00H〜07H(0FH):指定パレット番号
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し)

画面イメージの移動(INT AEH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
[BX+00H]←上下に移動するドット数(W)
[BX+02H]←左右に移動するドット数(W)
[BX+04H]←クリアフラグ
00H:移動後の領域をパレット0に設定
01H:移動後の領域をバックグラウンドカラーに設定
(返)AH←終了条件(00H:正常終了 05H:不正呼び出し)

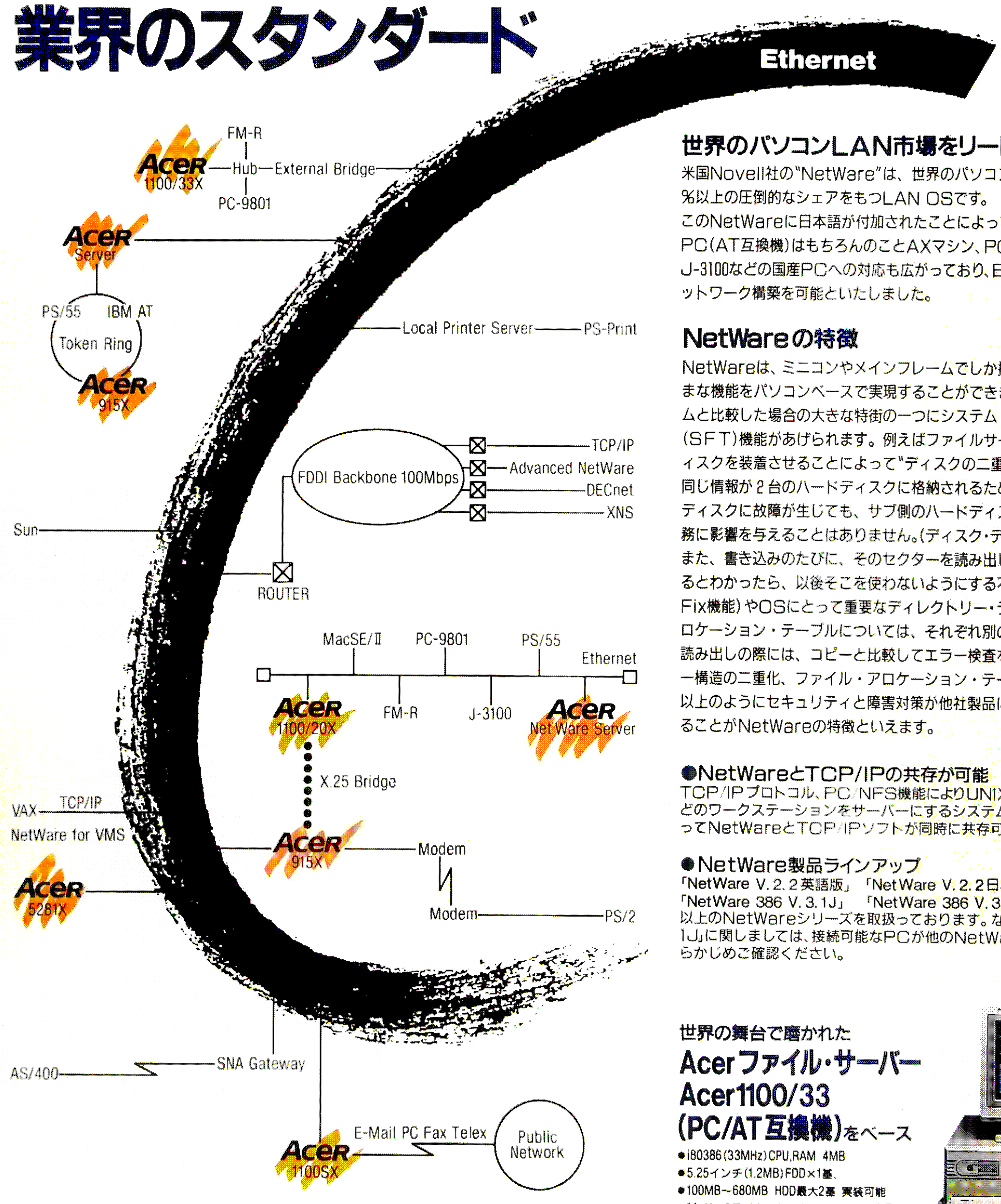
指定ドットのパレット番号取得(INT AFH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
BX←パラメータリスト先頭のオフセットアドレス
ES←DS
[BX+00H]←パレット番号を求める点の X 座標(W)
[BX+02H]←パレット番号を求める点の Y 座標(W)
(返)AH←終了条件(00H:正常終了)
AL←指定されたドットのパレット番号

画面イメージの展開(INT CEH)
(呼)DS←グラフLIOワークエリアのセグメントアドレス
SS/SP←グラフLIOスタックエリア
AX←指定領域左上点 X 座標(0000H〜027FH)
BX←指定領域左上点 Y 座標(0000H〜018FH)
CL←指定領域 X 方向ドット数(00H〜FFH)
CH←指定領域 Y 方向ドット数(02H/82H, 04H/84H, 08H)
DI ←格納域オフセットアドレス(0000H〜FFFFH)
ES←格納域セグメントアドレス(0000H〜FFFFH)
(返)AH←不定

Net Ware

Acer
The Word for Value

パソコンLANにおける業界のスタンダード



THE ARCHITECTURE EXTENDED-AX

世界のパソコンLAN市場をリード

米国Novell社の"NetWare"は、世界のパソコンLAN市場において60%以上の圧倒的なシェアをもつLAN OSです。
このNetWareに日本語が付加されたことによってACERなどのIBM-PC(AT互換機)はもちろんのことAXマシン、PC-9801、FM-R、PS-55、J-3100などの国産PCへの対応も広がっており、日本語・英語両環境でのネットワーク構築を可能といたしました。

NetWareの特徴

NetWareは、ミニコンやメインフレームでしか提供できなかったさまざまな機能をパソコンベースで実現することができます。他のLANシステムと比較した場合の大きな特徴の一つにシステム・フォルト・トレラント(SFT)機能があげられます。例えばファイルサーバーに2台のハードディスクを装着させることによって"ディスクの二重化"が可能であり、常に同じ情報が2台のハードディスクに格納されるため、万一メインのハードディスクに故障が生じても、サブ側のハードディスクが動作し続け通常業務に影響を与えることはありません。(ディスク・デュープレックス機能)
また、書き込みのたびに、そのセクターを読み出し、セクターに不良があるとわかったら、以後そこを使わないようにする不良セクター対策(Hot Fix機能)やOSにとって重要なディレクトリー・テーブルとファイル・アロケーション・テーブルについては、それぞれ別の場所に書き込んでおき、読み出しの際には、コピーと比較してエラー検査をします。(ディレクトリー構造の二重化、ファイル・アロケーション・テーブルの二重化など…)
以上のようにセキュリティと障害対策が他社製品に比べて非常に強力であることがNetWareの特徴といえます。

●NetWareとTCP/IPの共存が可能

TCP/IPプロトコル、PC/NFS機能によりUNIXマシンおよびSUNなどのワークステーションをサーバーにするシステムをサポート。これによってNetWareとTCP/IPソフトが同時に共存可能になりました。

●NetWare製品ラインアップ

「NetWare V.2.2英語版」「NetWare V.2.2日本語版」
「NetWare 386 V.3.1J」「NetWare 386 V.3.11英語版」
以上のNetWareシリーズを取扱っております。なお「NetWare 386 V3.1J」に関しましては、接続可能なPCが他のNetWareと異なりますのであらかじめご確認ください。

世界の舞台で磨かれた
Acerファイル・サーバー Acer1100/33 (PC/AT互換機)をベース

- i80386(33MHz)CPU,RAM 4MB
- 5.25インチ(1.2MB)FDD×1基、
- 100MB～680MB HDD最大2基 実装可能
- 14インチモノクロモニター、モニターI/Fカード
- 16ビットバス用 Ethrnetカード1枚 標準装備
- フロアスタンド他

※Net WareはNOVELL Inc.の登録商標です。

丸文株式会社
東京都中央区日本橋大伝馬町8-1 丸文ダイヤビル 〒10 第2営業本部 営業第3部 PC&ネットワークグループ TEL (03)3639-9897(ダイヤルイン)

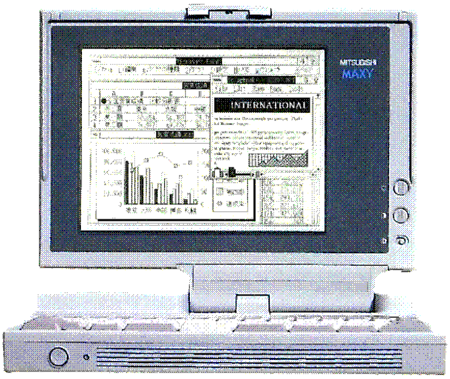
〈資料請求番号 P03〉

MITSUBISHI

技術がつくる高度なふれあい SOCIO-TECH

片手に乗った、スーパービジネス。

ライバルに差をつける、32ビットノートパソコン。

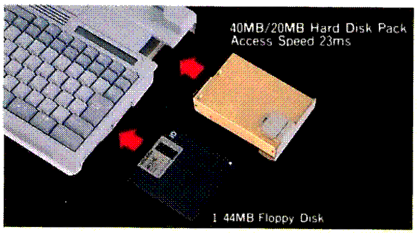


スーパービジネスは今、MAXYNOTE386から始まる。

「時代の最先端を切り開くビジネスエリートのために。」スーパービジネスに対応する、MAXYNOTE386の誕生です。A4ファイルサイズのコンパクトボディに、高速32ビットCPU386™SX(16MHz)を搭載。さらにノート初の高速演算プロセッサ387™SXも実装可能(オプション)。広範なビジネスの高速処理を実現します。

着脱式大容量ハードディスクパックで、思いのままに、スーパービジネス。

片手でポンッと出し入れ自在、フロッピー感覚の超小形2.5インチハードディスクパック(40MB/20MB)を内蔵。大容量情報を効率的に、思い通りにコントロールできます。さらに、統合操作環境MS-Windows、統合ソフトMAXYMATEなどを標準装備。ビジネスパワーを高める即戦力ツールとして即座に活用できます。



MAXYSTATIONⅡで、さらに拡がる、スーパービジネス。

本体一体形のコンパクトな拡張ユニットMAXYSTATIONⅡをオプションで用意。PC/ATフルサイズカードが2枚利用可能です。また、100MBハードディスクも内蔵可能。PC/ATの豊富なハードウェア資産を活用した、多彩なネットワーク構築など、デスクトップに迫る豊かな拡張力を提供します。

●高性能CPU386™SX(16MHz)搭載、387™SX(オプション)●RAM標準1MB(最大5MB)●FDD 3.5インチ×1●HDD 2.5インチ、交換自在なハードディスクパック方式(40MB/20MB)●非同期内蔵モデム実装可(オプション)●RS-232C×1(最大2)●PC/ATフルサイズカード2枚実装可(オプション)●パワーONで瞬時に画面を再現するレジューム機能●MS-DOS、MS-Windows、MAXYMATE、MAXYメニューシステムを標準装備

AX do Next

世界標準機・三菱AXパソコン
MAXYNOTE386

標準価格 ●B12(HDD 20MBモデル)398,000円(税別) ●B14(HDD 40MBモデル)438,000(税別)〈この他に省スペースデスクトップタイプもあります〉

ニーズから発想する三菱情報通信システム Think&Link

AX THE ARCHITECTURE EXTENDED-AX

MS-DOS、MS-Windowsは米国マイクロソフト社、386™SX、387™SXは米国インテル社、PC/ATは米国IBM社、MAXY、MAXYSTATION、MAXYMATEは三菱電機㈱の登録商標です。MAXYメニューシステムは、㈱エー・ジー・ソフトのAZ-MOS(エージーモス)をベースとしたメニューシステムです。AZ-MOSは㈱エージーソフトと㈱アイシーエムが開発したインストール方式「IBFファイル(インストールバッチファイル)システム」を採用しています。
世界標準化を推進するAX。日本のソフトウェアメーカ、ハードウェアメーカ、流通各社によるAX協議会★のもと、OS、キーボード、画面、プリンタ部を規格化。国内・海外の豊富なソフトや周辺機器の共通利用を実現します。★1990年12月現在参加553社

三菱電機株式会社

お問合せは…三菱電機株式会社 〒100 東京都千代田区丸の内2-2-3(三菱電機ビル) 本社情報処理機器事業部計画部(03)3218-3285 本社情報処理機器部(03)3218-3515 本社システム営業第一部(03)3218-3547 本社システム営業第二部(03)3218-3298 本社情報通信システム計画部(03)3218-3236 本社電機事業部(03)3218-3158 ●札幌(011)212-3739 ●仙台(022)264-5656 ●大宮(048)653-0261 ●千葉(0472)41-8671 ●横浜(045)212-3521 ●新潟(025)241-7220 ●金沢(0762)52-8821 ●名古屋(052)565-3246 ●大阪(06)347-2421 ●広島(082)248-5330 ●高松(0878)25-0009 ●福岡(092)721-2346

カタログ請求券 AXパソコン プログラマーズ 9109 あなたの電話番号をご記入ください

T4911777709989 〈資料請求番号P04〉 雑誌コード17777-09